НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра автоматики та управління в технічних системах
(повна назва кафедри)

«На правах рукопису»
УДК _____

«До захисту допущено»
Завідувач кафедри
_____ _____
(підпис) (ініціали, прізвище)
“___”_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 121 «Інженерія програмного забезпечення»
(код і назва)

на тему: Моделі і методи швидкого створення веб застосувань

Виконав: студент 6 курсу, групи ІТ-71МН
(шифр групи)

Вовк Євгеній Андрійович
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник д.т.н., професор Теленик С.Ф.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент д.т.н., професор Бідюк Петро Іванович
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»

Факультет (інститут) Факультет інформатики та обчислювальної техніки
(повна назва)

Кафедра автоматики та управління в технічних системах
(повна назва)

Рівень вищої освіти – другий (магістерський)
(код і назва)

Спеціальність 121 «Інженерія програмного забезпечення»
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ _____
(підпис) (ініціали, прізвище)
«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Вовк Євгеній Андрійович
(прізвище, ім'я, по батькові)

1. Тема дисертації Моделі і методи швидкого створення веб застосувань
науковий керівник дисертації д.т.н., професор Теленик С.Ф.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «02» квітня 2019 р. №877-с

2. Строк подання студентом дисертації 14 травня 2019

3. Об'єкт дослідження Моделі створення веб-застосувань

Предмет дослідження Моделі, методи та інформаційна технологія розробки веб застосувань

5. Перелік завдань, які потрібно розробити

Аналіз проблеми автоматизації створення застосувань, огляд існуючих рішень, розробка математичної моделі, розробка алгоритму автоматизації створення веб-застосувань

6. Орієнтовний перелік ілюстративного матеріалу

Схема опрацювання запиту веб-додатком, діаграма станів обробки запиту, загальний вигляд бізнес процесу ,архітектура інформаційної технології, модель трирівневої архітектури, схема бази даних, діаграма класів системи, діаграма компонентів.

7. Орієнтовний перелік публікацій

«Adaptive system of automatic control», «WINTER INFOCOM ADVANCED SOLUTIONS 2018»

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання 02 квітня 2019

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка

Студент _____ _____
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ _____
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Магістерська дисертація на здобуття ступеня магістру на тему “ Моделі і методи швидкого створення веб-застосувань”: 125с., 33 рис., 22 табл., 2 додатки, 26 джерел.

Об'єкт дослідження – моделі створення веб-застосувань.

Мета роботи – підвищення швидкодії створення веб застосувань.

У магістерській дисертації розглядається проблема використання формальних засобів для швидкого створення ефективних web-додатків одного класу. Пропонується концептуальний підхід до її вирішення на основі аналізу особливостей побудови web-додатків. Підхід базується на визначенні стандартної архітектури web-додатків і виборі її складових з використанням формальних методів відповідно до вимог користувача. Наводиться формальна логічна система з використанням якої проектування web-додатків здійснюється як процес виведення заданої відповідно до потреб користувача формули, який визначає схеми виконання модулів системи. Важливою особливістю підходу є можливість 3D-візуалізації процесу проектування системи, що створює умови для ефективної взаємодії розробників і машинних інструментів розробки.

ВЕБ-ЗАСТОСУВАННЯ, ПАТТЕРН, ШАБЛОНІЗАЦІЯ КОДУ, АРХІТЕКТУРНА ДЕКОМПОЗИЦІЯ.

Aster's thesis for master's degree on the topic "Models and methods of rapid web application creation": 125p, 33 figures, 22 tables, 2 annexes, 26 sources.

Object of study – web application development models.

The purpose of the work – increasing the speed of web application creation.

The problem of the rapid creation of effective web-applications of one class with using formal means are considered. The conceptual approach to its solution is offered on the basis of analysis of the peculiarities of web-applications construction. The approach is based on defining the standard web application architecture and selecting its components using formal methods in accordance with user requirements. A formal logical system is used, which uses the design of web-applications as a process of outputting the formula specified according to the needs of the user, which defines the schemes of execution of the modules of the system. An important feature of the approach is the ability to 3D-visualize the process of designing the system, which creates conditions for effective interaction between developers and machine development tools.

WEB-APPLICATIONS, PATTERNS, CODE GENERATION, APPLICATION ARCHITECTURE.

ЗМІСТ

ПЕРЕЛІК ВИКОРИСТАНИХ СКОРОЧЕНЬ

СУБД – система управління базами даних

ІТ – інформаційні технології

API - application programming interface

ІТС – інформаційно-телекомунікаційні системи

СУБД – система управління базами даних

ITIL- IT Infrastructure Library

ITSM -IT Service Management

ПЗ – програмне забезпечення

XML - eXtensible Markup Language

JSON - JavaScript Object Notation

SQL -Structured Query Language

БД – база даних

ANSI - American national standards institute

СУРБД - система управління реляційними базами даних

MVC - Model-View-Controller

JSP - Java Server Pages

HTTP - HyperText Transfer Protocol

ASP - Active Server Pages

S.O.L.I.D. – single responsibility, open-closed, Liskov substitution, interface segregation, dependency inversion

JDBC - Java DataBase Connectivity

JMX - Java Management Extensions

DNS - Domain Name System

LINQ - Language Integrated Query

REST - Representational State Transfer

ОС – операційна система

DOM - Document Object Model

ООП – об’єктно орієнтоване програмування

HTML - HyperText Markup Language

CSS - Cascading Style Sheets

AJAX - Asynchronous Javascript and XML

SOA – сервісно орієнтована архітектура

ВСТУП

Створення інформаційних систем сьогодні здійснюється на основі сучасних методологічних концепцій, які успадкували найважливіші ідеї класичних методологій типу SADT,IDEFO, одночасно збагативши їх новими ідеями.

Найперспективнішим напрямом розвитку програмних продуктів є клієнт серверна взаємодія в більшості побудована на основі HTTP протоколу, тобто веб застосування, що включають в себе ту саму обробку даних, подаючи її в зручному для користувача вигляді.

Формалізація досить специфічно представлена в сучасних методологіях. У класичних методологіях її розглядали як основу автоматизації. По суті формалізація тоді уявлялася як невід'ємною здатність кожної технології, яка реалізувала методологію, інструмент автоматичного продукування складових інформаційної системи, насамперед програмного забезпечення, баз даних, на підставі їх формальних описів. Сьогодні формалізація більше пов'язана з описанням архітектури системи і описи переважно використовується для постановки задач учасникам проекту, обговорення проміжних результатів, визначення наступних кроків.

Але є реальна можливість надати формалізації більш важливої ролі в сучасних методологіях створення інформаційних систем. По-перше, цьому сприяє стандартизація архітектури і компонентів інформаційних систем. По-друге, у рамках стандартизованих архітектур напрацьовано багато компонентів, які можуть бути використані для проектування нових систем. По-третє, розвиток математичної логіки і штучного інтелекту останніми роками забезпечив як інструменти формального описання систем, що проектуються з точки зору «що має бути реалізоване», так і описання компонентів з точки зору «що вже реалізоване». По-четверте, з'явилися методології просторової візуалізації систем, що проектуються, які можна перенести на створення на основі компоненто-базованого підходу інформаційних систем.

Під час практики зроблено спробу використати напрацьовані формальні моделі і ефективні методи для побудови технології автоматизованого проектування інформаційних систем на основі сучасних методологій.

Оскільки ця проблема має колосальні масштаби ми виділили один із найбільш готових для реалізації компонент сучасних інформаційних систем. Мова йде про задачу розроблення, яка стає все більш популярною, а саме задачу розроблення систем з web-представленнями.

Метою та завданням є автоматизація розроблення систем з web-представленнями та відповідні моделі і методи швидкого створення веб застосувань.

Об'єктом дослідження: моделі створення веб-застосувань.

Предметом дослідження Моделі, методи та інформаційна технологія розробки веб застосувань.

КОНКРЕТНІ ЗАВДАННЯ роботи:

- аналіз проблеми створення застосувань
- огляд існуючих рішень
- розробка математичної моделі
- розробка алгоритму автоматизації створення додатків
- проведення експериментального дослідження.

МЕТОДОЛОГІЯ даної роботи базується на дослідженнях таких вчених як Теленик С.Ф., Єфремов К.В.

МАТЕРІАЛОМ для дослідження були сучасні фреймворки JHipster, Swagger, розглянуто ряд шаблонних рішень CMS таких як WordPress Joomla, SimplCommerce та інші.

НАУКОВА НОВИЗНА полягає в тому, що вперше було запропоновано рішення для автоматизації генерації коду та побудови додатків згідно з заданим бізнес-процесом.

АКУТАЛЬНІСТЬ роботи в тому, що не дивлячись на глобалізацію та стрімкий розвиток веб розробки, зокрема бізнес систем з веб представленням, доступні на даний час фреймворки та готові рішення не вирішують задачу

генерації коду для бізнес-процесу, а також для часткового вирішення даної задачі необхідна їх агрегація та додаткова обробка коду, що був згенерований. Під час роботи використовувався комплекс методів**:** загальні методи – вивчення інформаційних джерел з проблеми дослідження, аналіз сучасних розподілених систем обробки даних, математичне моделювання, описовий метод;

АПРОБАЦІЯ РОБОТИ. Результати виконаної дослідницької роботи були опубліковані в матеріалах до міжнародної науково-технічної конференції “ WINTER INFOCOM ADVANCED SOLUTIONS 2018” у 2018 році в м.Києві.

ПУБЛІКАЦІЇ. Результати виконаної дослідницької роботи були опубліковані у статті «Conceptual foundations of the use of formal models and methods for the rapid creation of web-applications» в сучасному науковому журналі «Adaptive system of automatic control» в 2019 році в м. Києві.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

На сьогоднішній день, значний об'єм ринку всіх програмних продуктів займають бізнес системи з веб представленням. При цьому від проекту до проекту змінюються дані з якими працюють розробники і користувачі, але не змінним залишається принцип їх реалізації. Доцільність автоматизації цього класу задач пояснюється значною трудомісткістю проектування і реалізації підзадач, які дублюються у кожному проекті, і при цьому власне зміні підлягають лише дані, з якими працюють, але не основні їх перетворення.

Як зазначалося вище, розробникам варто домовитися про те, що будь-яке застосування вибраного типу має в своїй основі напрацьовану архітектуру. Також легко домовитися про те, що будь-яке застосування вибраного типу має однаковий загальний вигляд і відрізняється від інших лише наявністю або відсутністю тих чи інших компонентів в залежності від вимог до функціоналу застосування.

Сьогодні при виборі будови застосувань доцільно дотримуватися принципів трирівневої архітектури. Відповідно у типовому застосуванні будемо виділяти представлення, бізнес-логіку і рівень доступу до даних. Кожен з яких відповідно до принципу декомпозиції на під задачі можна деталізувати на рівень агрегації класів що вирішують ту чи іншу задачу та називаються патерном. В кожному патерні виділяють абстрактну частину, що дає змогу ізолювати конкретну реалізацію від загальної поведінки та імплементацію, що надає відповідним класам абсолютно чітко визначені методи з їх сигнатурами і відповідною реалізацією. Детальніше декомпозиція полягає в структуруванні конкретного класу з наведеного патерну та доповнені його необхідними функціональними одиницями.

Розглянемо приклад детальніше кожний з рівнів архітектури.

1.1 Практичний приклад

Розглянемо приклад типової задачі, що включає в себе всі рівні архітектури та демонструє загальний вигляд системи. Необхідно реалізувати web систему з розмежуванням прав доступу відповідно з двома користувацькими ролями адміністратора та користувача. Користувач після логіну може завантажити файл з розширенням JSON в якому повинна міститись інформація що описує нове обладнання. Система повинна зберегти службову інформацію з файлу опрацювати її і вивести адміністратору на підтвердження. Згідно з вимогами до коду, а саме логіка опрацювання файлу може змінюватись код повинен бути чистим та гнучким, відповідати всім принципам S.O.L.I.D. З точки зору вимог по функціоналу система повинна вміти авторизувати користувачів, опрацьовувати, а саме перевіряти коректність інформації в завантаженому файлі, зберігати історію завантажень, та повідомляти адміністратора ресурсу через електрону пошту про нове не підтверджене завантаження файлу.

Виходячи з цього маємо наступну архітектуру.

- CommandInvoke.Db.Entities – збірка сутностей що відповідають таблицям в базі даних.
- CommandInvoke.Dal.Abstract абстракція з описом сигнатури CRUD операцій з джерела данних оперуючи сутностями з п.1.
- CommandInvoke.Dal.Impl.Ef – реалізація конкретної поведінки по роботі з MSSQL server засобами ORM Entity framework з використанням підходу Code first.
- CommandInvoke.Entities.Ui моделі якими оперує бізнес-логіка для видачі через Api
- CommandInvoke.Abstract абстракція з описом логіки, що покриває всі функціональні вимоги до системи
- CommandInvoke.Bl.Impl – реалізація конкретної поведінки по виконанню процесів системи

- CommandInvoke.Web – Rest API інтерфейс мережевої взаємодії з системою що працює з абстракцією бізнес-логіки і повертає дані для відображення

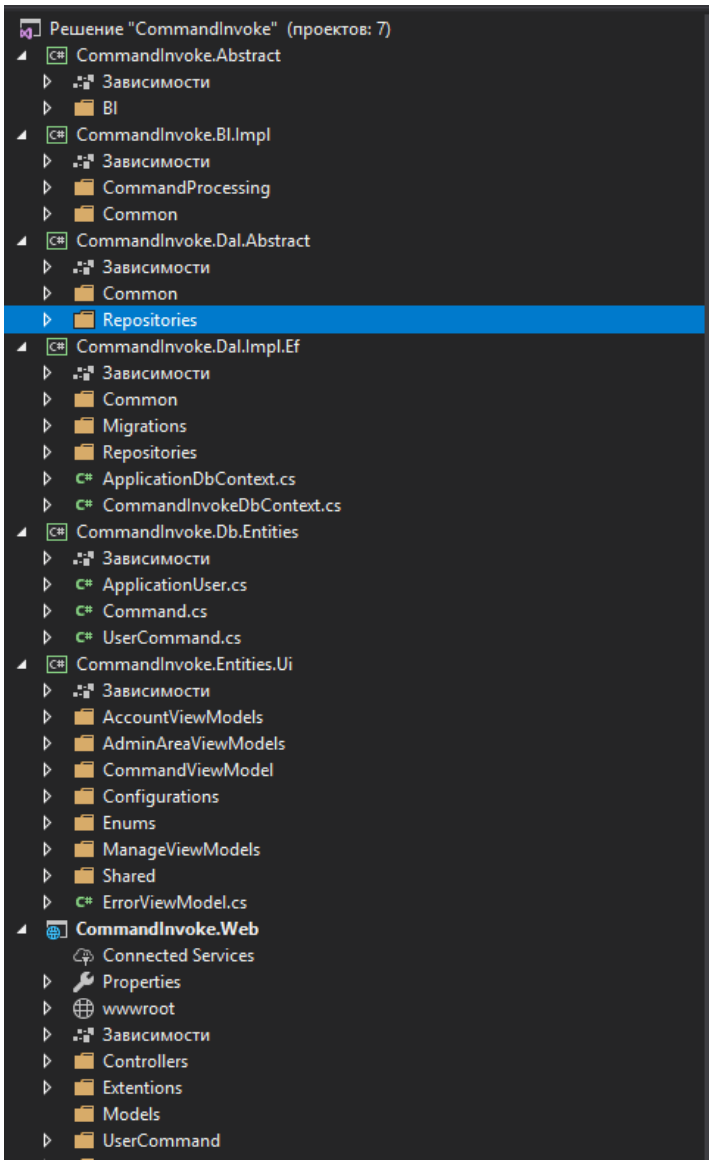

Рисунок 1.1 - Загальний вигляд готового рішення.

Розглянемо шаблони що застосовувались в ході реалізації даного рішення.

Для рівня доступу до даних використовувався патерн репозиторій. Для бізнес-логіки стратегія. І два шаблони рівнем вище котрі повинні в випадку шаблонізації бути задані як окремі модулі, що покривають бізнес-процес авторизації реєстрації, та відправку електронних повідомлень. Листинг програми наведений у Додатку Б – лістинг коду типової веб-системи. Як видно з опису задача з одного боку достатньо об’ємна та потребує часових затрат на

навчальному прикладі в розмірі 8 годин, а з іншого боку складається виключно з монотонної роботи по зберіганню та завантаженню даних з бази даних і більш складного процес їх обробки та двох регулярно використовуваних бізнес-процесів авторизації та відправки електронного листа.

1.2. Джерела даних

1.2.1 SQL

Структурована мова запитів (Structured Query Language -SQL) - це мова доступу до БД, що є стандартизованою та підходить для наступних СУРБД: Oracle, SQL Server, Sybase Access, MySQL.

В основі даної мови знаходиться стандарті ANSI. Основною сферою затосування є управління та доступ до сховища даних. Містяться відповідні команди, що дозволяють отримати доступ до БД та виконати стандартні операції з даними. До них відносять читання, запис, оновлення та видалення.

1.2.2 NoSQL

NoSQL ("not only SQL" - англ. не тільки SQL) - база даних, що базується на принципово іншому способі збереження даних. Ключовою відмінністю є відсутність типізації даних, що зберігаються. Тобто база даних оперує так званими документами що задаються в форматі JSON. Даний підхід забезпечує гнучкість даних що зберігаються. Розмежування на документи також дозволяє виконувати горизонтальне маштабування системи без ризику втрати консистентності даних. Також даний вид баз даних є гнучким до внесення коректив в моделі в ході доопрацювання системи. Поява та інтеграція сховищ даних подібного типу обумовлена сервісно-орієнтованою архітектурою та стрімким розвитком нетипізованих мов програмування.

1.2.3 REST API

REST (Representational state transfer) - це стиль архітектури програмного забезпечення для розподілених систем, як правило,

використовується для побудови веб-служб. REST є одним із способів реалізації архітектурного стилю «клієнт-сервер» - по суті, REST явно спирається на архітектурний стиль «клієнт-сервер». Кожна одиниця інформації однозначно визначається глобальним ідентифікатором, таким як URL. Кожна URL у свою чергу має строго заданий формат. Передача даних відбувається в тому ж вигляді, що й самі дані, без будь-яких прошарків.

Управління інформацією сервісу повністю грунтується на протоколі передачі даних. Для HTTP, наприклад, дія над даними задається за допомогою методів: GET (отримати), PUT (додати, замінити), POST (додати, змінити, видалити), DELETE (видалити). Таким чином, дії CRUD (Create-Read-Updtae-Delete) можуть виконуватися як з усіма 4-ма методами, так і тільки за допомогою GET і POST. Дане джерело даних може може фігурувати в випадках роботи зі сторонніми сервісами такими як Google maps, Azure Cloud і тд, так і при створенні власної сервісно орієнтованої архітектури.

SOA - Сервіс-орієнтована архітектура - модульний підхід до розробки програмного забезпечення, заснований на використанні розподілених, слабко пов'язаних замінних компонент, оснащених стандартизованими інтерфейсами для взаємодії за стандартизованими протоколами.

1.3 Робота з джерелами даних

Жодна сучасна система не може існувати без сховища даних, що відповідно потребує рівня комунікації з тим чи іншим сховищем даних в рамках програмного продукту. В якості сховища даних може виступати база даних, як SQL так і NoSQL, зовнішній ресурс в рамках роботи з зовнішнім інтерфейсом по обробці даних (API).

Також існує цілий ряд систем що використовують комбіноване сховище, декілька видів баз даних з додаванням зовнішнього ресурсу.

Відповідно в кожному з вище зазначених випадків можна чітко визначити перелік та алгоритм роботи зі сховищем.

1. Відкрити з’єднання

2. Створити запит
3. Направити запит до сховища
4. Отримати результат виконання запиту
5. Прочитати результат
6. Закрити з’єднання

Розглянемо даний алгоритм детальніше. Оскільки даний алгоритм є шаблонним необхідно конкретизувати моменти взаємодії в випадку кожного виду зовнішніх сховищ. Для баз даних, що NoSQL, що SQL алгоритм буде однаковим. Відповідно використовуємо рядок з’єднання в якому вказана адреса серверу та порт, ім’я бази даних, інформація про часовий ліміт на встановлення з’єднання та налаштування безпеки для з’єднання, а також інформацію про тип авторизації і інформацію про користувача, що необхідна для логіну в сервер баз даних. Відповідно по TCP встановлюємо з’єднання з сервером баз даних використовуючи адресу серверу та вказаний порт, авторизуємось в сервері баз даних та отримуємо доступ до вказаної бази даних. Як було зазначено вище будуємо запит певного типу з переліку CRUD (Create-Read-Updtae-Delete) механізм та мова побудови запиту залежить від способу реалізації взаємодії, але в будь якому випадку на виході ми отримуємо запит на мові що підтримується сервером баз даних, відповідно до синтаксичних та лексичних вимог мови. По відкритому з’єднанню отримуємо результат виконання операції та данні. Після чого закриваємо з’єднання.

В випадку з REST API взаємодія відбувається по HTTP протоколу, готовий запит з відповідним тілом, заголовками (службовою інформацією) і відповідним типом Put, Post, Get, Delete, відправляється на кінцеву точку по URL читаємо готову інформацію, проте з’єднання вже буде зачиненим.

На основі вище наведеного алгоритму можна сказати що в незалежонсті від сутності та методу є ряд речей які повторюються. Так само як на наступному рівні композиції в незалежності від сутності повторюється загальна базова поведінка тобто абстракція даної поведінки є однаковою та зводиться до реалізації методів, що описують CRUD (Create-Read-Updtae-Delete) операції

для кожної сутності, а при необхідності містять методи по роботі з агрегатованими сутностями. Дані фактори в сукупності з розмежуванням додатку на шари, що спровоковане необхідністю гнучкості і породжують патерни для вирішення даної задачі. В ході виконання запиту і тестуванні патерне рішення надає гнучкості існуючому коду забезпечуючи єдину точку доступу до бази даних через визначений інтерфейс.

Існує два основних підходи, що розв’язують дану задачу:

Репозиторій - це фасад для доступу до бази даних. Весь код програми за межами сховища працює з базою даних через нього і тільки через нього. Таким чином, репозиторій інкасулює в собі логіку роботи з базою даних, це шар об'єктно-реляційного відображення в нашому додатку. Більш точно, репозиторій, або сховище, це інтерфейс для доступу до даних одного типу - один клас моделі, одна таблиця бази даних в простому випадку. Доступ до даних організовується через сукупність всіх репозиторіїв.

Unit Of Work - є класом обгорткою для репозиторію та являє собою контейнер для зберігання та створення репозиторіїв.

1.3.1 Repository

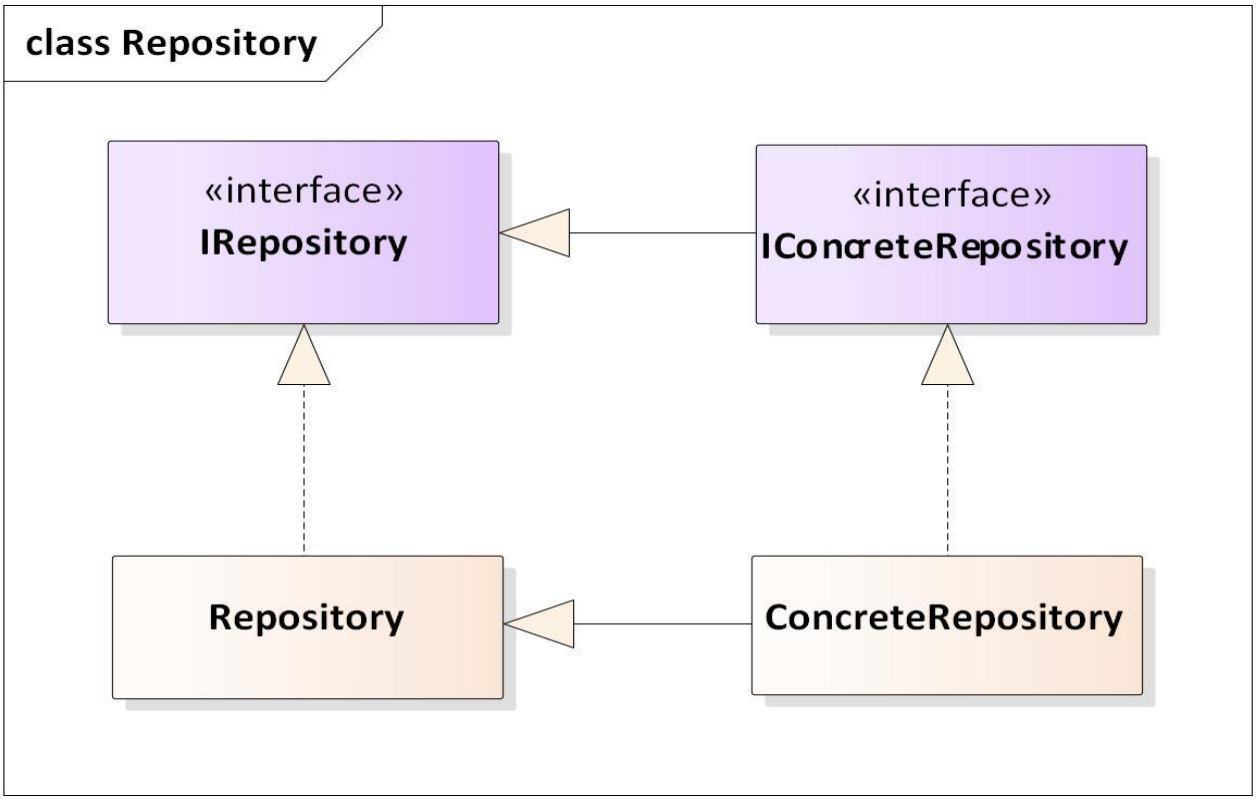

Рисунок 1.2- загальний вигляд патерну репозиторій

Шаблон "Repository", як видно з діаграми, складається з узагальненої абстракції і більш деталізованих її імплементацій. Розглянемо приклад шаблону з точки зору реалізації в коді.

- IRepository<T> шаблоний інтерфейс що описує базову поведінку для класів наслідників по роботі з даними;
- IEntityRepository більш конкретна абстракція задачею якої є зберігання в собі унікальних методів для роботи з поточною сутністю(Entity) та наслідується від IRepository<T> закладуючи абстракцію для базових методів але з чітко визначеним типом;
- BaseRepository<T> є класом що імплементує IRepository<T> та реалізовує всі його методи базові для всіх сутностей і містить привязку до певної конкретно вибраної технології по роботі з БД;
- EntityRepository конкретний клас наслідни що імплементує свій конкретний інтерфейс (IEntityRepository) та унаслідує базовий клас

BaseRepository<T> для реалізації стандартних методів по роботі з даними.

Тобто роботу з даними визначаємо у традиційний для об’єктно-орієнтованого програмування спосіб. По-перше з кожною сутністю пов’язуємо певну узагальнену поведінку, яка притаманна всім елементам рівня доступу до даних. По-друге доповнена абстракція для кожного елемента рівня доступу до даних розширює його поведінку, додаючи унікальні методи для поточної сутності.

Відповідно конкретний клас сховища успадковує свій інтерфейс, отримуючи базові методи загальні для всіх і свої методи з власного інтерфейсу, а також успадковує клас, який реалізовує базову поведінку, за фактом чого містить реалізацію тільки своїх методів.

Як можна побачити з опису, паттерн має визначену архітектуру і є розширюваним за рахунок засобів додавання нових класів і інтерфейсів для роботи з кожною сутністю. А також патерн Repository є посередником між шаром області визначення і шаром розподілу даних, працюючи, як звичайна колекція об'єктів області визначення. Об'єкти-клієнти створюють опис запиту декларативно і направляють їх до об'єкта-сховища (Repository) для обробки. Об'єкти можуть бути додані або видалені з сховища, як ніби вони формують просту колекцію об'єктів. А код розподілу даних, прихований в об'єкті Repository, подбає про відповідних операціях в непомітно для розробника. У двох словах, патерн Repository инкапсулирует об'єкти, представлениие в сховище даних і операції, що здійснюються над ними, надаючи більш об'єктно-орієнтоване уявлення реальних даних.

1.3.2 Unit Of Work

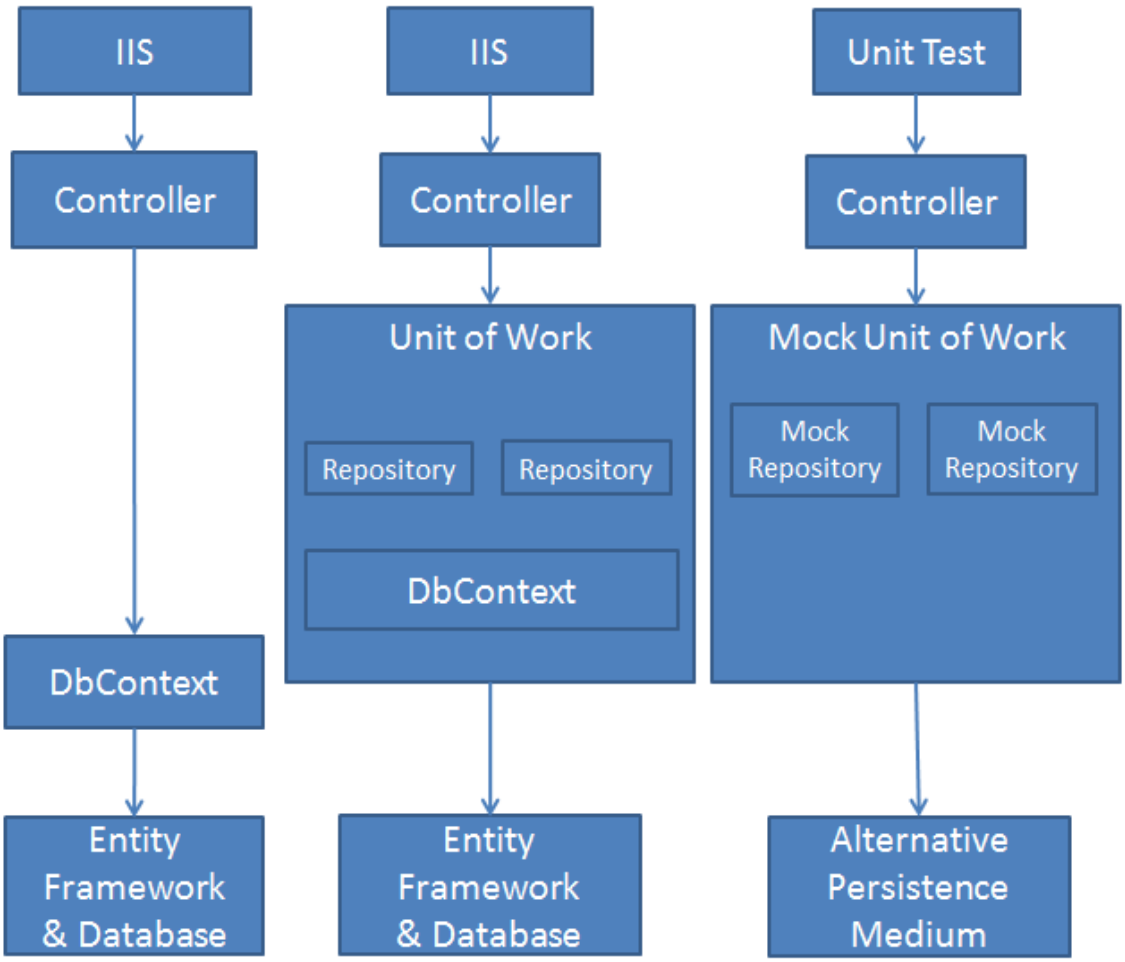

Рисунок 1.3 – загальний вигляд патерну Unit of work

Unit Of Work є класом обгорткою для репозиторію та являє собою контейнер для зберігання та створення репозиторіїв.

Дана надбудова над репозиторієм застосовується в випадку коли в додатку відсутня DI (Dependency Injection) як механізм підстановки імплементацій за інтерфейсом.

Як можна побачити з опису, паттерн має визначену архітектуру і є розширюваним за рахунок засобів додавання нових класів і інтерфейсів для роботи з кожною сутністю.

1.4 Бізнес-логіка

Бізнес-логіка переважно полягає у обробленні даних, отриманих з рівня доступу до даних, шляхом використання поведінкових шаблонів, таких як стратегія, також класів поведінки, які описують логіку оброблення даних, відхиляючись від канонічних шаблонів, в певних випадках команд. Кожний складовий компонент реалізує вже кінцеву функціональність, яку очікує отримати користувач. Тобто присутній перехід між сутністю що описує таблицю бази даних і сутністю що виходить в ході опрацювання бізнес-логікою. Даний процес є посередником між рівнями області визначення і розподілу даних (domain and data mapping layers), використовуючи інтерфейс, схожий з колекціями для доступу до об'єктів області визначення. Система зі складною моделлю області визначення може бути спрощена за допомогою додаткового рівня, наприклад Data Mapper, який би ізолював об'єкти від коду доступу до БД. У таких системах може бути корисним додавання ще одного шару абстракції поверх шару розподілу даних (Data Mapper), в якому б був зібраний код створення запитів. Це стає ще більш важливим, коли в області визначення безліч класів або при складних, важких запитах. У таких випадках додавання цього рівня особливо допомагає скоротити дублювання коду запитів. Згідно закладеної концепції складові компоненти є розширюються за рахунок засобів додавання конкретних класів з вузько націленою поведінкою.

1.4.1 Паттерн стратегії

Паттерн стратегія спрямований на реалізацію різної поведінки в залежності від компоненту, який знаходиться рівнем вище і звертається до даної стратегії.

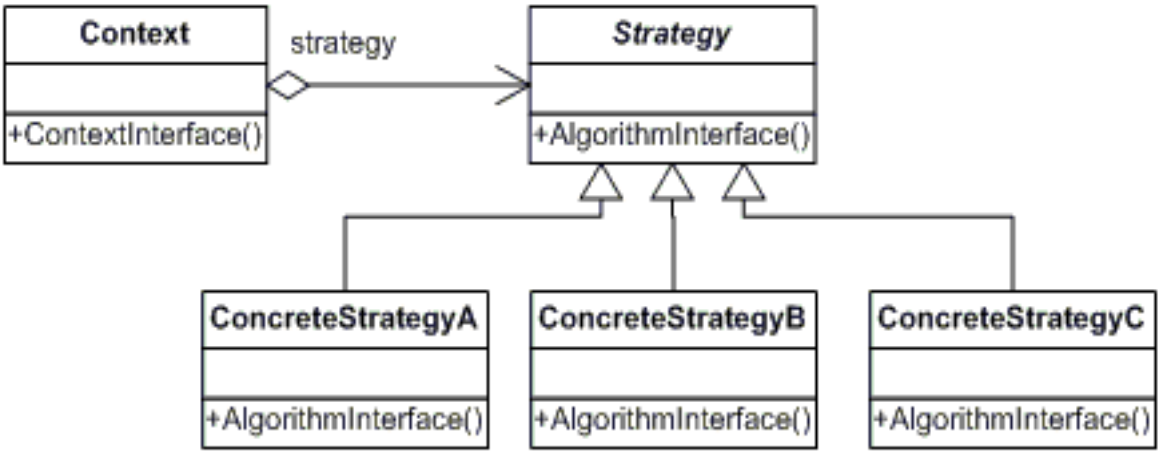

Рисунок 1.4 - загальний вигляд патерну стратегії

Як видно з діаграми присутній базовий опис стратегії з відповідним методом, який приймає на вхід необхідні параметри, і відповідно кілька реалізацій даного методу.

1.4.2 Паттерн команди

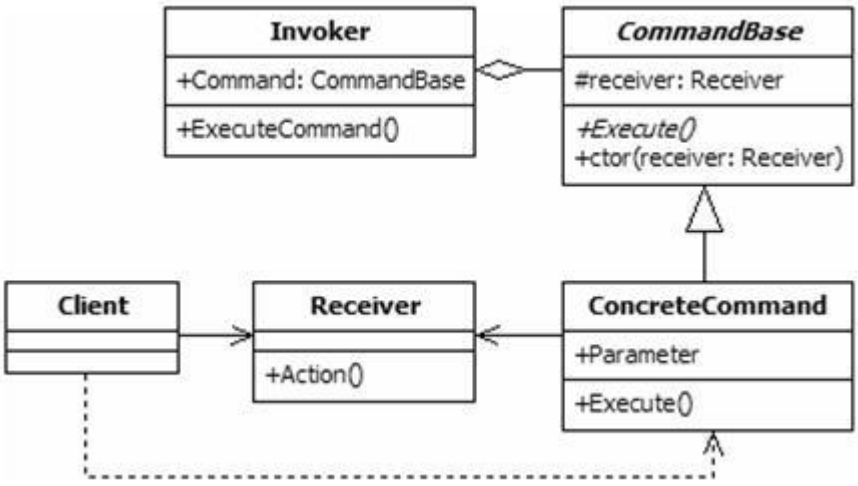

Рисунок 1.5. загальний вигляд патерну команди

Сфера застосування патерну команди полягає у здійсненні (виконанні) тієї чи іншої логічної поведінки на вимогу, дозволяючи абстрагуватися від конкретної логіки самої команди всередині механізму.

Ще один механізм, який застосовується, - це інтерфейс з описами поведінки і клас реалізації, що містить елементи обробки даних і формування

результату, необхідного для компонента, що знаходиться в архітектурі рівнем вище.

1.4.3 Паттерн ланцюг обов’язків

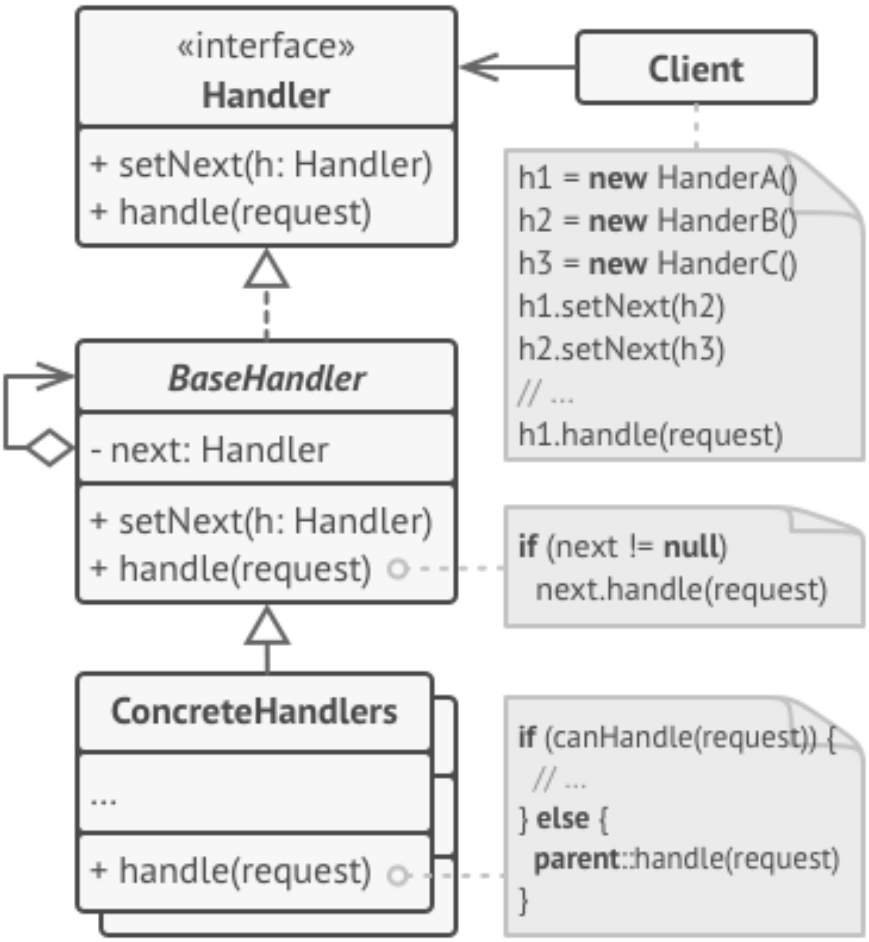

Рисунок 1.6 - загальний вигляд патерну ланцюг обов’язків

Оброблювач визначає загальний для всіх конкретних оброблювачів інтерфейс. Зазвичай достатньо описати єдиний метод обробки запитів, але може мати місце оголошення і метод виставлення наступного обробника. Тобто в ході делегування методу до кінцевої точки за рахунок обходу ланцюжку викликів і передачі відповідальності за рішення до конкретного компоненту якій здатен правильно опрацювати даний запит та на якому припиняється подальший обхід ланцюга .

Даний паттерн застосовується в випадку необхідності динамічного вибору обробника поточної події.

1.4.4 Паттерн посередник

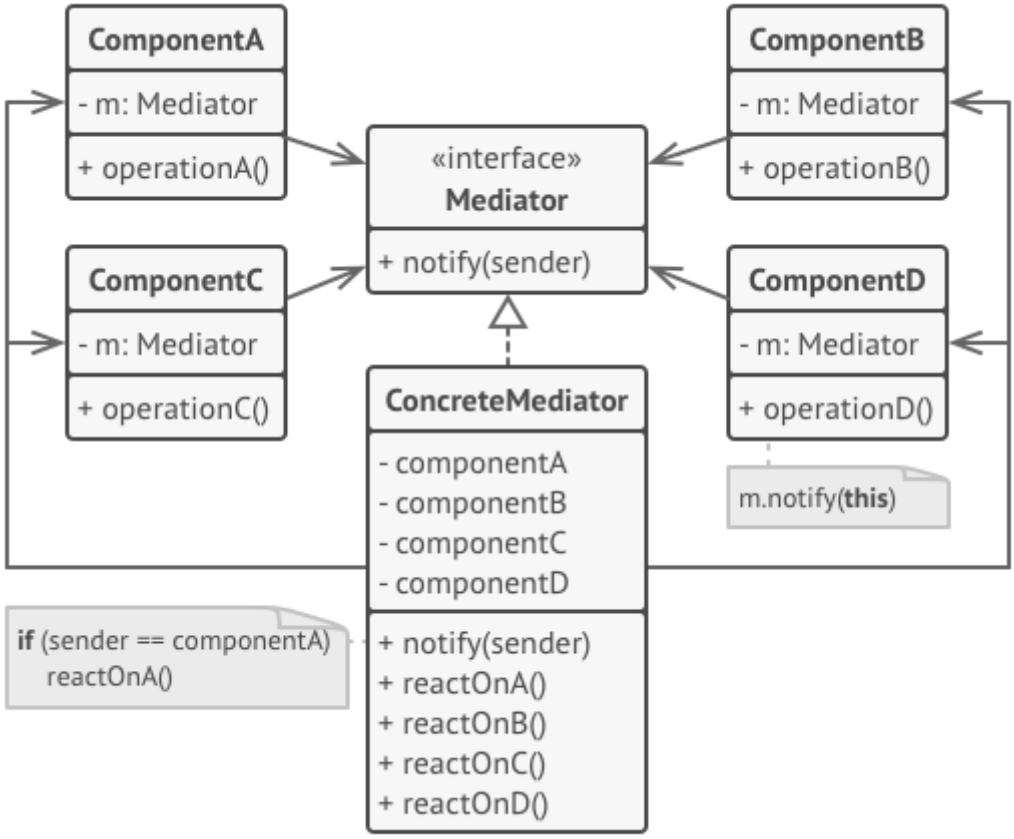

Рисунок 1.7 - загальний вигляд патерну посередник

Компоненти - це різноманітні об'єкти, що містять бізнес-логіку програми. Кожен компонент зберігає посилання на об'єкт посередника, але працює з ним тільки через абстрактний інтерфейс посередників. Завдяки цьому, компоненти можна повторно використовувати в іншій програмі, зв'язавши їх з посередником іншого типу.

Посередник визначає інтерфейс для обміну інформацією з компонентами. Зазвичай вистачає одного методу, щоб повідомляти посередника про події, що сталися в компонентах. В параметрах цього методу можна передавати деталі події: посилання на компонент, в якому вони є і будь-які інші дані.

Конкретний посередник містить код взаємодії кількох компонентів між собою. Найчастіше цей об'єкт не тільки зберігає посилання на всі свої компоненти, а й сам їх створює, керуючи подальшим життєвим циклом.

Компоненти не повинні взаємодіяти один з одним безпосередньо. Якщо в компоненті відбувається важлива подія, він повинен сповістити свого посередника, а той сам вирішить - чи стосується подія інших компонентів, і чи варто їх сповіщати. При цьому компонент-відправник не знає, хто обробить його запит, а компонент-одержувач не знає, хто його надіслав.

Даний паттерн вирішує задачу абстрагування від жорстких зав’язків та довгих ланцюгів виклику.

1.4.5 Паттерн стан

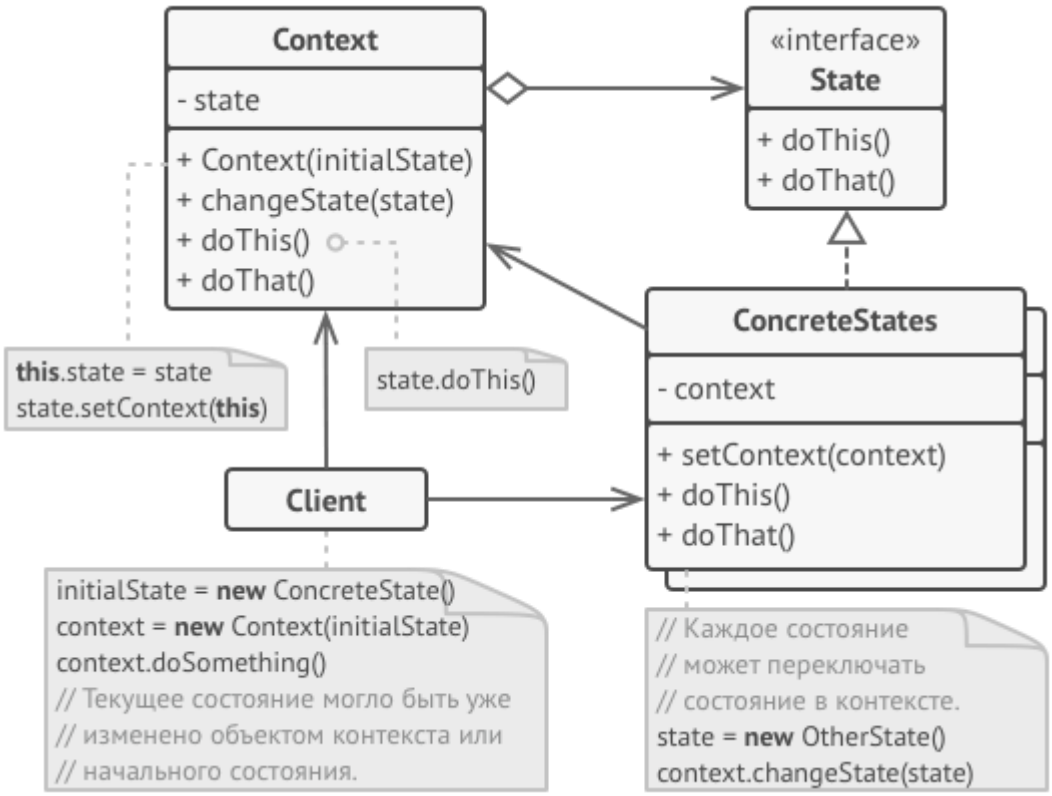

Рисунок 1.8 – загальний вигляд патерну стан

Контекст зберігає посилання на об'єкт стану і делегує йому частину роботи, яка залежить від станів. Контекст працює з цим об'єктом через загальний інтерфейс станів. Контекст повинен мати метод для присвоювання йому нового об'єкта-стану.

Стан описує загальний інтерфейс для всіх конкретних станів.

Патерн Стан пропонує створити окремі класи для кожного стану, в якому може перебувати об'єкт, а потім винести туди поведінки, які відповідають цим станам.

Замість того, щоб зберігати код всіх станів, початковий об'єкт, званий контекстом, буде містити посилання на один з об'єктів-станів і делегувати йому роботу, залежну від стану.

1.5 Представлення

Для реалізації представлень тобто зовнішньої точки доступу до додатку також використовуються патерни, з ціллю розмежування зовнішнього вигляду та логіки поведінки тієї чи іншої сторінки. Основним патерном, що вирішує дану задачу є MVC (Model-view-controller). Розгляне детальніше даний паттерн на основі платформи .net.

Стаття MVC Overview від Майкрософт дає таке визначення MVC: Model-view-controller (MVC, «Модель-уявлення-контролер») - схема використання декількох шаблонів проектування, за допомогою яких модель даних програми, користувальницький інтерфейс і взаємодія з користувачем розділені на три окремих компонента так, що модифікація одного з компонентів надає мінімальний вплив на інші. Схема роботи MVC представлена на рисунку 1.9.

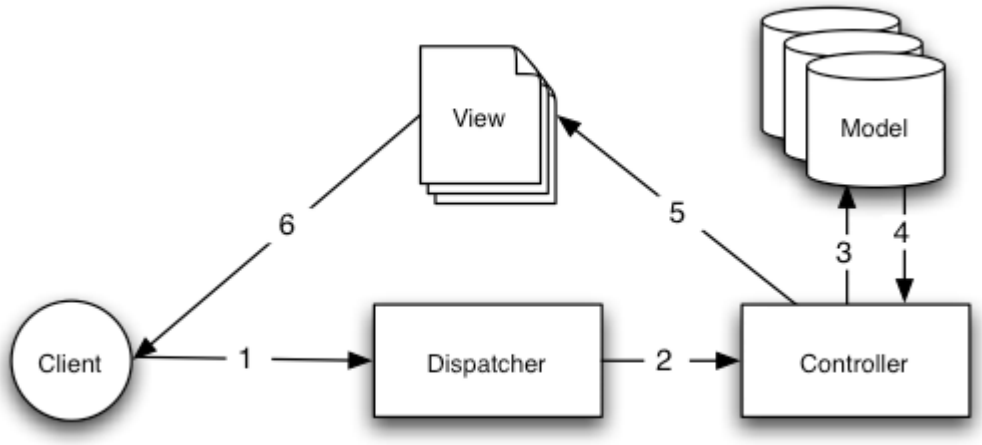

Рисунок 1.9 - схема роботи MVC.

До складу платформи MVC входять наступні компоненти:

- Моделі. Об'єкти моделей є частинами програми, що реалізують логіку для домену даних програми. Об'єкти моделей часто отримують і зберігають стан моделі в базі даних. У невеликих додатках ця модель має на увазі концептуальний, а не фізичний поділ. Наприклад, якщо програма тільки зчитує набір даних і відправляє його в уявлення, то фізичний шар моделі і пов'язаних класів відсутня. У цьому випадку набір даних приймає роль об'єкта моделі.
- Подання. Уявлення служать для відображення інтерфейсу програми. Інтерфейс користувача зазвичай створюється на основі даних моделі.
- Контролери. Контролери здійснюють взаємодію з користувачем, роботу з моделлю, а також вибір подання, що відображає користувальницький інтерфейс. У додатку MVC подання лише відображають дані, а контролер обробляє дані, що вводяться і відповідає на дії користувача. Наприклад, контролер може обробляти рядкові значення запиту і передавати їх у модель, яка може використовувати ці значення для відправки запиту до бази даних.

Шаблон MVC дозволяє створювати додатки, різні аспекти яких (логіка введення, бізнес-логіка і логіка інтерфейсу) розділені, але досить тісно взаємодіють один з одним. Інтерфейс користувача розташовується в поданні. Логіка введення розташовується в контролері. Бізнес-логіка знаходиться в моделі. Це поділ дозволяє працювати зі складними структурами при створенні програми, тому що забезпечує одночасну реалізацію тільки одного аспекту. Наприклад, розробник може сконцентруватися на створенні подання окремо від бізнес-логіки.

Два основних типи заснованих на MVC додатків:

1. З пасивної моделлю - модель не має жодних способів впливати на уявлення або контролер, і використовується ними в якості джерела даних для відображення. Всі зміни моделі відслідковуються контролером і він же відповідає за перемальовування подання, якщо це необхідно. Така модель частіше використовується в структурному програмуванні, так як в цьому випадку модель являє просто структуру даних, без методів їх обробних.

2. З активною моделлю - модель оповіщає уявлення про те, що в ній відбулися зміни, а уявлення, які зацікавлені в оповіщенні, підписуються на ці повідомлення. Це дозволяє зберегти незалежність моделі, як від контролера, так і від уявлення.

Платформа ASP.NET MVC надає такі можливості:

- Поділ завдань програми (логіка введення, бізнес-логіка і логіка користувацького інтерфейсу), широке можливості тестування і розробки на основі тестування. Всі основні контракти платформи MVC засновані на інтерфейсі і підлягають тестуванню за допомогою макетів об'єкта, які імітують поведінку реальних об'єктів програми. Додаток можна піддавати модульним тестування без запуску контролерів у процесі ASP.NET, що прискорює тестування і робить його більш гнучким. Для тестування можливе використання будь-якої платформи модульного тестування, сумісної з. NET Framework.
- Розширюють і доповнюють платформа. Компоненти платформи ASP.NET MVC можна легко замінити або налаштувати. Розробник може підключати власний механізм уявлень, політику маршрутизації URL-адрес, серіалізацію параметрів методів дій і інші компоненти. Платформа ASP.NET MVC також підтримує використання моделей контейнера впровадження залежності (DI) і інверсії елемента управління (IOC). Модель впровадження залежності дозволяє впроваджувати об'єкти в клас, а не очікувати створення об'єкта самим класом. Модель інверсії елемента

управління вказує на те, що якщо один об'єкт вимагає інший об'єкт, то перші об'єкти повинні отримати другий об'єкт із зовнішнього джерела (наприклад, з файлу конфігурації). Це полегшує тестування.

- Розширена підтримка маршрутизації ASP.NET. Цей потужний компонент зіставлення URL-адрес дозволяє створювати додатки з зрозумілими URL-адресами, які можна використовувати в пошуку.URL-адреси не повинні містити розширення імен файлів і призначені для підтримки шаблонів іменування URL-адрес, що забезпечують адресацію, оптимізовану для пошукових систем (SEO) і для передачі репрезентативного стану (REST).
- Підтримка використання розмітки в існуючих файлах сторінок ASP.NET (ASPX), елементів управління (ASCX) і головних сторінок (MASTER) як шаблонів уявлень.Разом з платформою ASP.NET MVC можна використовувати існуючі функції ASP.NET, наприклад вкладені головні сторінки, вбудовані вирази (<% =%>), декларативні серверні елементи управління, шаблони, прив'язку даних, локалізацію і т. д.

Підтримка існуючих функцій ASP.NET. ASP.NET MVC дозволяє використовувати такі функції, як перевірка справжності за допомогою форм і Windows, перевірка достовірності за URL-адресою, членство і ролі, кешування виводу і даних, управління станом сеансу і профілю, спостереження за працездатністю, системою конфігурації та архітектурою постачальника.

MVC спрощує структуру, розбиваючи програму на три окремих модуля. Не використовує view state і server-based форми, проте складніше в коддінге ніж звичайні Веб форми. По суті, є альтернативою концепцією побудови додатків, однак, враховуючи вищесказане, не представляє інтересу в нашому випадку.

Виходячи з детального опису патерну можна говорити про те, що контролер виступає з’єднувачем безпосередньо бізнес-логіки і представлення.

1.6 Огляд існуючих рішень

Спроби автоматизувати створення web-застосувань робляться досить давно.

Звичайно ж варто згадати про вже готові рішеннях подібні до CMS. Логіка поведінки даних конструкторів полягає у використанні готової системи на весь спектр завдань за рахунок засобів вибору виключно теми графічного оформлення і додавання або відключення тих чи інших сторінок. Вона суттєво відрізняється від описаної вище логіки, що тягне за собою низку недоліків, які, в інтересах подальшого викладення матеріалу, доцільно розглянути докладніше.

Перший недолік пов'язаний з технологією реалізації CMS. Переважне використання PHP до того ж без розмежування рівнів архітектури не супроводжується використанням шаблонів. Це тягне за собою присутність величезного пласту НЕ читабельного і важко редагованого коду.

Другим недоліком роботи системи, яка будується для виконання всього функціоналу і налаштовується шляхом включення і вимикання опцій, є маленька швидкість. Це є наслідком того, що система завантажує окремо кожен компонент, який часто є надлишковим для даного сайту, оскільки побудований за шаблоном, розрахованим на весь функціонал. Зрозуміло, що це негативно позначається на продуктивності.

На рисунку 1.9 наведені порівняльні дані продуктивності різних засобів на прикладі реалізації бізнес-процесу оформлення замовлення. На графику відображена швидкість відгуку системи в залежності від кількості користувачів.

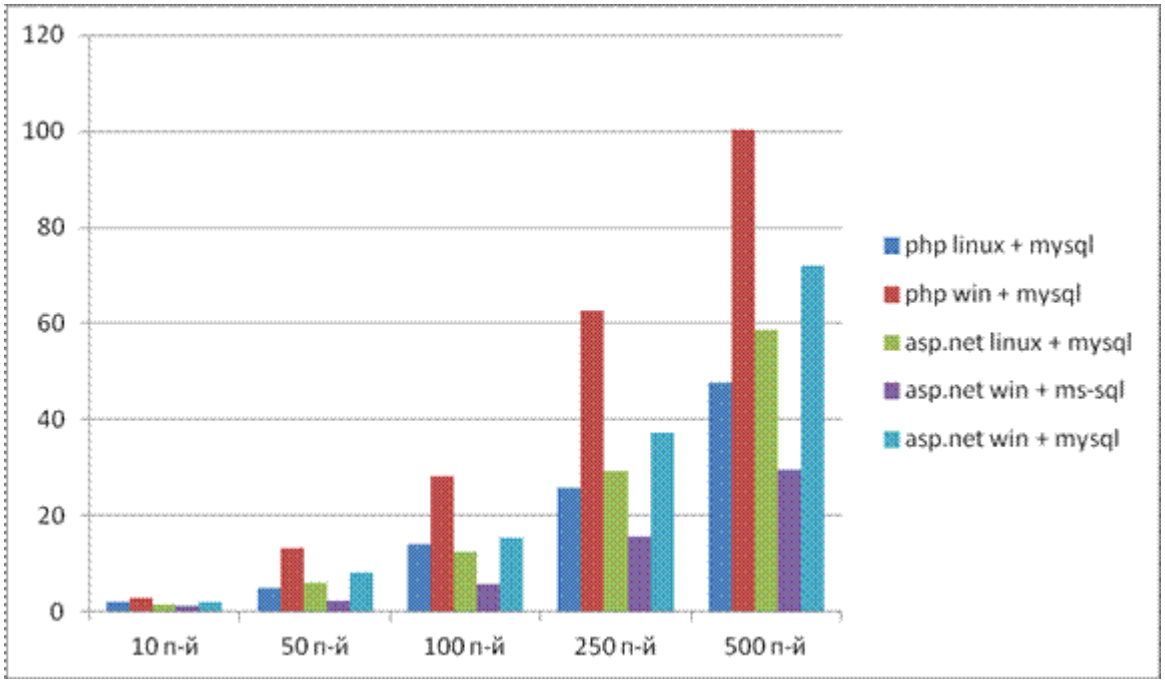

Рисунок 1.10 - порівняльні дані продуктивності

Але найголовнішим недоліком є відсутність налаштування системи на різних рівнях. Про це побіжно було згадано вище, але зараз розглянемо цю проблему докладніше. Використовуючи CMS, ми не можете впливати на структуру бази даних, на кількість коду, який забезпечує роботу з нею, створювати свої власні поведінкові елементи на рівні бізнес-логіки. Ми можемо впливати тільки на зовнішній вигляд і включення або відключення вже готових і запропонованих функціональних можливостей. Наслідком такого рішення є низька продуктивність, яка супроводжується відсутністю можливості будувати шаблонно складні функціональні одиниці, а також не читабельність і складність ручного корегування існуючого коду.

На рисунку 1.9 відображено відмово стійкість при навантаженні на наборі технологій, відповідно дана характеристична діаграма відображає аргументацію вибору технології для реалізації комерційних рішень для високопродуктивних систем і аргументує не здатність CMS комплексно вирішувати поставленні задачі та відображає недоліки швидко дії безкоштовної бази даних My sql.

Ще одним рішенням є композиція автоматизації окремих модулів за рахунок використання фреймворків на тій чи іншій технології. Розглянемо дану альтернативу детальніше. Оскільки систему можна подати як композицію модулів, кожен модуль, як композицію проектів, кожен проект, як композицію класів, кожен клас, як композицію методів і тд. Під автоматизацію ряду модулів існують вже готові фреймворки по генерації коду. Недоліками даного підходу є складність самого процесу шаблонізації в рамках всього рішення, оскільки шаблонізація модулю досить проста задача а агрегація набору шаблонізованих модулів вимагає від програміста теж значних часових затрат через не консистентність версій фреймоврків, що використовуються в одному та іншому модулі, через відсутність достатньої кількості опцій для детального налаштування процесу шаблонізації і в багатьох випадках необхідність власноруч редагувати код.

1.7 Постановка проблеми

Відповідно до вище зазначеного можна сказати що ключовою проблемою розробки на поточний момент часу є або великі затрати на штат програмістів які в будь-якому разі будуть зіштовхуватись з написанням шаблонного коду, що вимагає серйозних часових затрат на повний цикл розробки починаючи від процесу формалізації вимог закінчуючи процесом тестування даного коду, або ж шаблонізація окремо взятих модулів як складова частина першого рішення, що відповідно приводить до необхідності редагування коду і виправлення тих самих проблем при композиції шаблонізованих модулів, або ж використання готових конструкторів для додатків які не містять компонентів, що дозволяють реалізовувати високопродуктивні задачі в наслідок не коректності технологій реалізації даних систем та загального підходу з реалізацією уже готового продукту в якій можна вносити мінімальні зміни.

1.8 Висновок

Виходячи з аналізу існуючих рішень та наведеного прикладу була сформована наукова актуальність роботи, поставлено завдання для дослідження та сформовано проблематику, що є підґрунтям для наукової актуальності. Ключовим аспектом аналізу є невідповідність систем на кшталт CMS для генерації веб застосунків з урахуванням можливості управління процесом автоматизації на кожному з рівнів та відповідної можливості інтеграції та задання тих чи інших правил. Відповідною альтернативою при більш детальному розгляді архітектури веб додатків є шаблонізація в момент їх реалізації на кожному з під рівнів архітектури за допомогою фреймворків що розроблені під ту чи іншу технологію. При більш детальному розгляді очевидно, що рівень доступу до даних на основі сутностей з бази даних піддається шаблонізації, як окрема складова, аналогічно можна шаблонізувати створення REST API, про те рівень бізнес-логіки не містить можливостей для шаблонізації оскільки потребує детального опису, а архітектурний рівень фреймворків не дає їм можливості містити достатньо гнучкості в рамках налаштувань. Ще одна з ключових проблем, що постає перед програмістом в випадку застосування даного підходу – комбінування генерованих елементів в загальний додаток, що потребує внесення відповідних змін в відповідний авто згенерований код продукту.

Відповідно до результатів аналізу, було встановлено проблему дослідження та дана точка є відправною для створення рішення, що міститиме переваги рішень, а саме можливість шаблонізації процесу створення додатку з можливістю детального налаштування даного процесу. В поточному розділі було аргументовано загальний підхід декомпозиції глобальної задачі на незалежні одиниці, що піддаються шаблонізації та наведено огляд основних концепцій побудови додатків.

2 КОНЦЕПЦІЯ АВТОМАТИЗАЦІЇ РОЗРОБЛЕННЯ

2.1 Основні поняття генерації коду додатку

Задачам генерації коду(шаблонізації) притаманна декомпозиція на ряд підзадач, які дублюються у кожному проекті. При цьому від проекту до проекту змінюються дані з якими працюють розробники і користувачі, але не змінним залишається принцип їх реалізації. Доцільність автоматизації цього класу задач пояснюється значною трудомісткістю проектування і реалізації підзадач, які дублюються у кожному проекті, і при цьому власне зміні підлягають лише дані, з якими працюють, але не основні їх перетворення.

Як зазначалося вище, розробникам варто домовитися про те, що будь-яке застосування вибраного типу має в своїй основі напрацьовану архітектуру. Також легко домовитися про те, що будь-яке застосування вибраного типу має однаковий загальний вигляд і відрізняється від інших лише наявністю або відсутністю тих чи інших компонентів в залежності від вимог до функціоналу застосування.

Сьогодні при виборі будови застосувань доцільно дотримуватися принципів трирівневої архітектури. Відповідно у типовому застосуванні будемо виділяти представлення, бізнес-логіку і рівень доступу до даних. Кожен з яких відповідно до принципу декомпозиції на під задачі можна деталізувати на рівень агрегації класів що вирішують ту чи іншу задачу та називаються патерном. В кожному патерні виділяють абстрактну частину, що дає змогу ізолювати конкретну реалізацію від загальної поведінки та імплементацію, що надає відповідним класам абсолютно чітко визначені методи з їх сигнатурами і відповідною реалізацією. Детальніше декомпозиція полягає в структуруванні конкретного класу з наведеного патерну та доповнені його необхідними функціональними одиницями.

2.2 SOA - Сервіс-орієнтована архітектура

В випадку використання зовнішнього сервісу як джерела данних необхідно розглянути загальну сервісно орієнтовану архітектуру.

SOA - Сервіс-орієнтована архітектура – що представляє собою композицію модулів, як підхід до розробки програмного забезпечення, що заснований розподілених, не звязаних жорстко та замінних компонентах, що оснащенні стандартизованими інтерфейсами для взаємодії за стандартизованими протоколами.

Основні принципи SOA:

- Стандартизований метод комунікації - сервіси мусять бути підпорядкованими стандарту комунікації, що прийнятий колективно.
- Слабкий зв'язок – сервіси мусять бути незалежними однин від одного, і у разі припинення роботи одного, тобто виходу з ладу сервісу, інші сервіси мусять працювати в штатному режимі.
- Повторне використання сервісів – розприділення логіки по сервісах для наявності можливості їх багаторазового використання в межах сервісної архітектури.
- Абстрактність сервісу - крім опису сигнатури концевих точок сервісу за допомогою сервіс-контракту, він приховує внутрішню реалізацію бізнес-логіки від зовнішнього світу.
- Автономність – логіка інкапсульована в сервісі підконтрольна тільки йому.

Переважна більшість технологій та протоколів взаємодії сервісів, побудовані відповідно до принципів SOA, незалежать від мови написання сервісу, що забезпечує, надання користувачам можливості комбінувати сервіси різних типів в розподілених системах, та будувати єдиний шаблон опису сервісів в якості універсальної сполучної ланки.

Структура SOA відображена на рисунку.

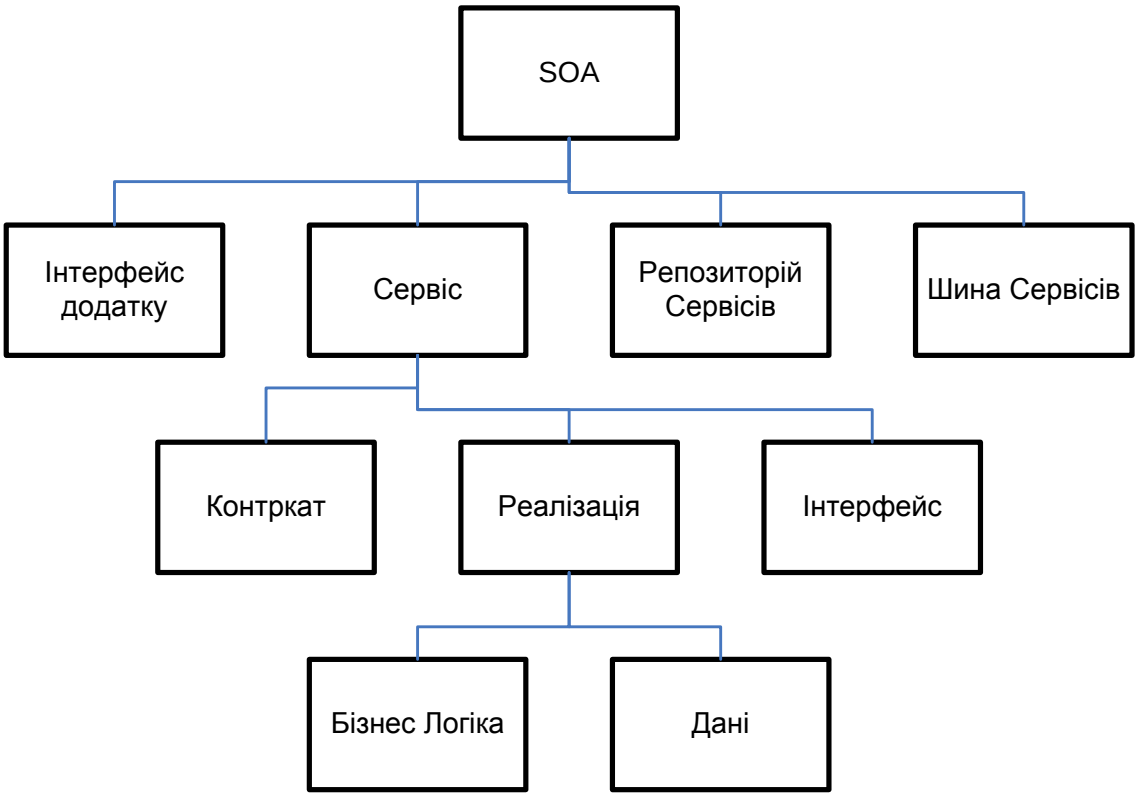

Рисунок 2.1 - струткту́ра SOA.

Веб-сервіси є базою для створення додатків, функціональність яких можне бути використане за допомогою стандартних протоколів Інтернет. Концепція веб-сервісів може бути побудована за допомогою ряду технологій, стандартизація яких вказана в World Wide Web Consortium (W3C).

Веб-сервіси – один з варіантів побудови компонентної архітектури.

XML - фундамент для створення переважної більшості технологій, пов'язаних з веб-сервісами.

Для створення концепції віддаленої взаємодії додатків і відповідних користувачів з веб-сервісами використовується Simple Object Access Protocol (SOAP) [4]. SOAP забезпечує комунікацію в рамках розподілених систем, абстрагуючись від об'єктної моделі, операційної системи або мови програмування. Дані передаються за допомогою визначених форматів між сервісної взаємодії, одним з яких є XML документ. Для SOAP сервісу він містить особливий формат(спосіб задання) згідно якому має бути заданий

документ. Існує ряд альтернатив рішенню для взаємодії з веб-сервісами (CORBA, WCF та ін.).

Відповідно до визначення W3C, веб-сервіси це додатки, що доступні по відповідним протоколам, які є стандартизованими для мережі Інтернет. Не існує вимог, стосовно використання веб-сервісами якогось конкретного транспортного протоколу. Специфікація SOAP регламентує, яким чином вибудовується зв'язок повідомленнями SOAP і відповідний протокол передачі.

Частіше всього передача SOAP повідомлень реалізується по протоколу HTTP. Також має місце використання наступних транспортних протоколів, як SMTP, FTP, TCP для вирішення даної задачі.

Відповідно до оприділення W3C, "WSDL - формат XML для задання однозначного опису мережних сервісів в якості набору кінцевих операцій, робота яких здійснюється за допомогою повідомлень, що містять документно-орієнтовану або процедурно-орієнтовану інформацію". Документ WSDL містить повний опис інтерфейсу веб-сервісу із зовнішнім світом. WSDL файл це документ у форматі XML, що найменування методів, що будуть надані веб-сервісом, відповідні параметри даних методів, їх назви та типи, і адресу Listener `а сервісу. В своїй сутності, він є чітко визначеним за загальними принципами взаємодії прошарком, що надає можливість використання сервісів, створених на базі різних платформ.

WSDL містить все необхідне для опису веб-сервісів, включаючи:

- URL адресу сервісу
- Механізми міжсервісної взаємодії, котрі розуміє веб-сервіс
- Функції, тобто набір методів, які можуть бути виконані веб-сервісом
- Структуру повідомлень веб-сервісу

Приклад WSDL файлу:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="ServiceName"
targetNamespace="http://ServiceUrlSample/ServiceName/"
xmlns:tns="http://ServiceUrlSample/ServiceName/"
```

```
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlnsɔwsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <message name="SampleGetMethod">
    <part name="Result" type="xsd:string" />
  </message>
  <message name="SampleGetRequest">
    <part name="prefix" type="xsd:string" />
  </message>
  <portType name="GuidPortType">
    <operation name="getGuid" parameterOrder="prefix">
      <input message="tns:SampleGetRequest"/>
      <output message="tns:SampleGetMethod" />
    </operation>
  </portType>
  <binding name="GuidEinding" type="tns:GuidPortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="getGuid">
      <soap:operation soapAction="urn:ServiceName#SampleGetRequest"/>
      <input>
        <soap:body use="encoded" namespace="urn:ServiceName"
        encodingstyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
      <output>
        <soap:body use="encoded" namespace="urn:ServiceName"
        encodingstyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  </binding>
  <service name="GuidService"
    <port name="GuidPort" binding="tns:GetBinding">
      <soap:address location="http://localhost/php/ServiceName.php"/>
    </port>
  </service>
</definitions>
```

Більшість платформ самостійно здатні формувати WSDL описи сервісів, забезпечуючи розробнику відповідну можливість роботи з сервісом як із звичайним класом.

Технологія Universal Description, Discovery and Integration (UDDI) забезпечує підтримку ведення реєстру веб-сервісів. В наслідок підключення до цього реєстру, споживачеві надається можливість знаходження веб-сервісів, які є найкращим рішенням відповідно до його потреб. Технологія UDDI забезпечує людигу чи програмний клієнт можливостями публікації та пошуку необхідного сервісу. Публікація і пошук в реєстрі надається програмі-клієнтові як відповідний набір веб-сервісів реєстру UDDI [5].

Веб-сервіси розцінюються, як програмне забезпечення проміжного шару. Використання веб-сервісів доступне як клієнтським програм, які безпосередньо взаємодіють з користувачем, так і іншим додаткам (у тому числі й іншим веб-сервісам).

Розробниками концепції веб-сервісів запропоновано наступні сценарії застосування веб-сервісів:

1. Веб-сервіси в якості реалізації бізнес-логіки додатка. Тобто, бізнес-логіка нового додатку, буде поміщена у веб сервіс, що в свою чергу розташовано в сервісі застосувань (див. рис. 2.2).

Рисунок 2.2 - сценарій застосування сервісу як реалізації логіки додатку.

2. Веб-сервіси як засіб інтеграції. Тобто, застосування веб-сервісу як способу відаленого доступу для клієнтів до внутрішньої інформаційної системи компанії, або в якості взаємодії компонента (наприклад, EJB, COM-компонента) з різними віддаленими клієнтами (див. рисунок 2.3).

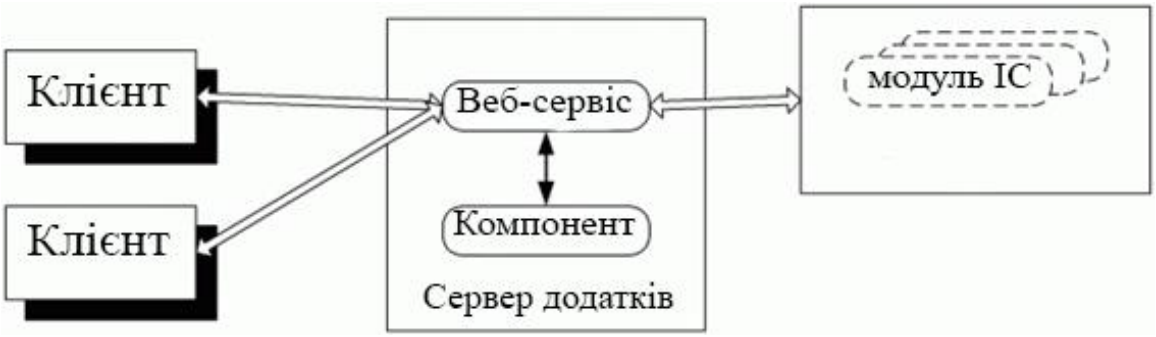

Рисунок 2.3 - сценарій застосування сервісу як засобу інтеграції

3. Створення альтенативного сервісу. В даному випадку, при створені нового веб-сервісу буде використано опис інтерфейсу веб-сервісу, що вже існує. Відповідно, сервіс містить велику кількість потенційних клієнтів з моменту початку роботи, а комунікація з ним стороніх сервісів та клієнтів не вимагає істотних змін.

Концепція веб-сервісів містить в собі можливість ведення реєстру веб-сервісів. З такого реєстру можна отримати опис інтерфейсу. Після впровадження і відповідно створення нового сервісу, має сенс помістити його в реєстр. Тоді при пошуку сервісів клієнти, що реалізують відповідний інтерфейс, отримають посилання і на новий веб-сервіс.

4. Застосування веб-сервісу, як складової частини при створенні програми.

Веб-сервіси можуть бути застосовані додатоком як зовнішні компоненти, що надають чітко визначену та вузько направлену функціональність, відповідно до потреб клієнта, матеріальних та часових обмежень і рівня складності завдання, що забезпечує системі велику гнучкість (див. рисунок 2.4).

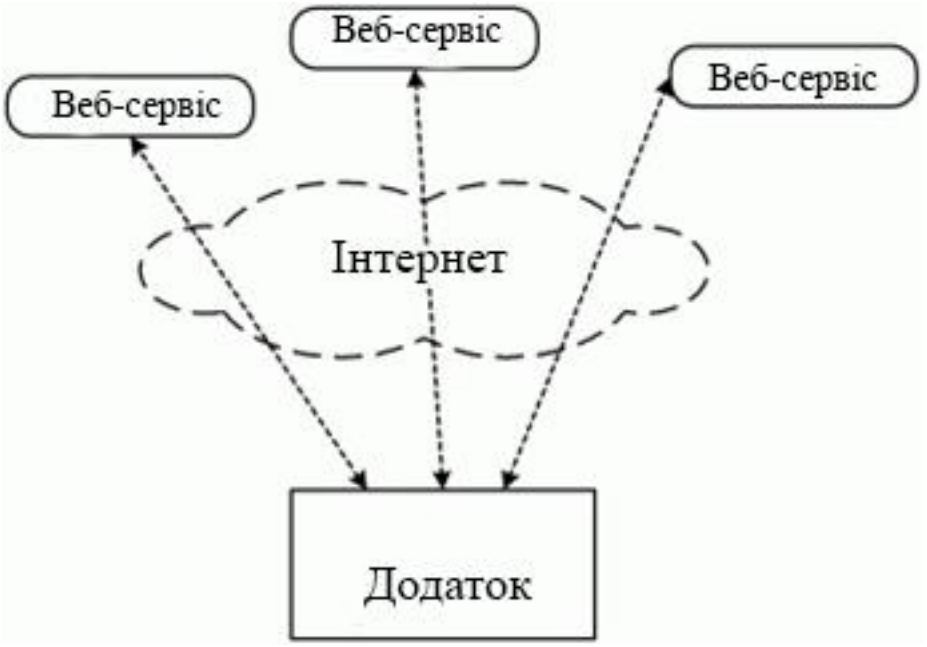

Рисунок 2.4 - застосування сервісу як зовняшнього структурного блоку програми.

Окрім цього, має місце і інші варіанти застосування веб-сервісів. Прикладом цього є дослідження з використання веб-сервісів в якості елементів для побудови розподілених обчислень та відповідного класу інформаційних систем, до яких належать однорангові, так і зі системи з складною структурою.

Для застосування веб-сервісу клієнту необхідно володіти такою інформацією:

- адреса розташування веб-сервіс;
- ім'я веб-сервісу;
- сигнатура методу, що використовуватиметься.

У найпростішому випадку дана інформація може бути задана при розробці клієнта, що проілюстровано рисунком.

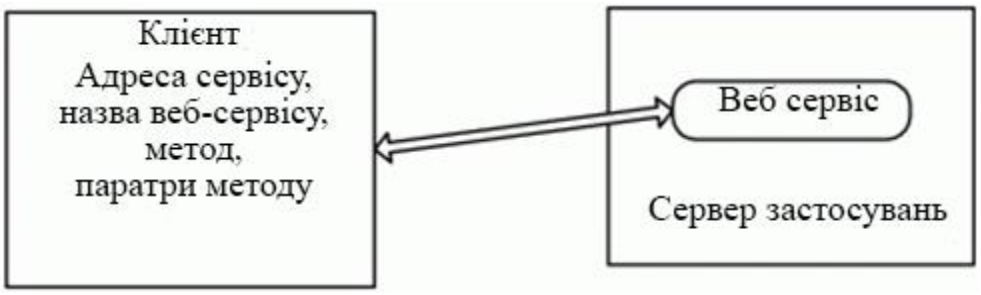

Рисунок 2.5 - інформація для застосування веб-сервісу, що є задана при розробці клієнта.

Документ WSDL містить повний опис інтерфейсу веб-сервісу із зовнішнім світом, він надає клієнтові засіб для отримання необхідної інформації для подальшого застосування веб-сервісу (див. рисунок 2.6).

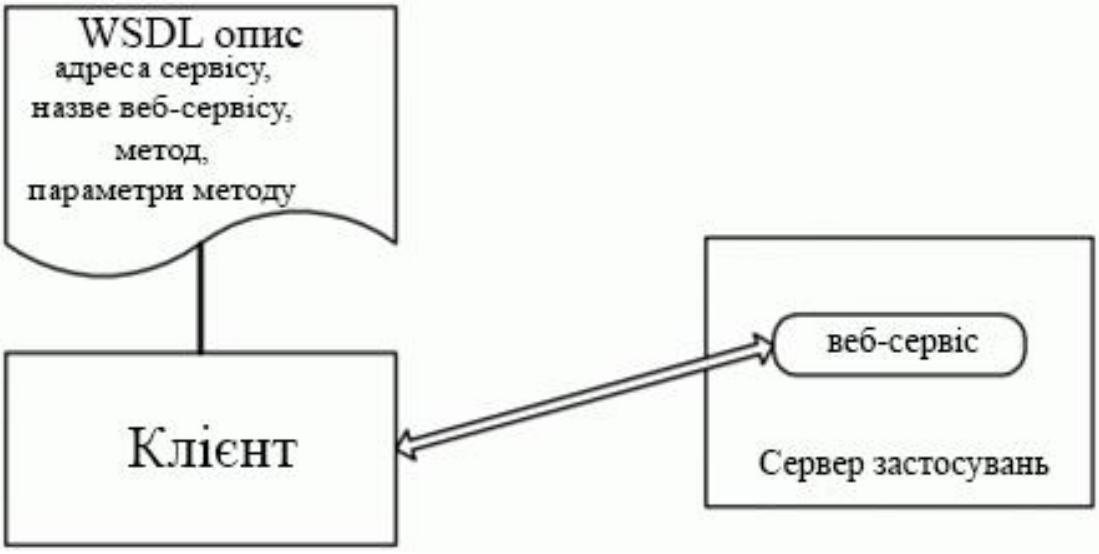

Рисунок 2.6 - WSDL опис інтерфейсу сервісу документом.

2.2.1 Оркестрація та Хореографія сервісів.

Виділяють два ключові підходи, для розв’язання проблеми координації роботи сервісів в розподілених системах:

Service orchestration (Приклад - BPEL) - метод композиції сервісів, з використанням сервісу-окрестартора, в один багатофункціональний бізнес-сервіс.

Web Service Choreography – метод, що регламентує інтерфейс взаємодії веб-сервісів між собою. Тобто кооперацію сервісів для виконання одного

глобального завдання, завдяки виконанню окремих його частини. Роль сервісу, визначає модель обміну повідомленнями з партнерами. Даний підхід є ефективним для простих завдань, але відповідно до зростання складності поставленого завдання та росту кількості задіяних сервісів стає громіздкою і неефективною.

2.3 Робота з джерелами даних

Виходячи з аналізу предметної області та наведеного опису способів роботи з джерелами даних маємо наступну послідовність дій :

- відкрити з’єднання;
- створити запит;
- направити запит до сховища;
- отримати результат виконання запиту;
- прочитати результат;
- закрити з’єднання;

А також загальний опис CRUD операцій і деталі їх реалізації. Розглянемо способи по роботі з даними детальніше вказавши ключові аспекти по автоматизації процесу створення даних шаблонів.

Існує два основних підходи, що розв’язують дану задачу:

Репозиторій - це фасад для доступу до бази даних. Весь код програми за межами сховища працює з базою даних через нього і тільки через нього. Таким чином, репозиторій інкасулює в собі логіку роботи з базою даних, це шар об'єктно-реляційного відображення в нашому додатку. Більш точно, репозиторій, або сховище, це інтерфейс для доступу до даних одного типу - один клас моделі, одна таблиця бази даних в простому випадку. Доступ до даних організовується через сукупність всіх репозиторіїв.

Unit Of Work - є класом обгорткою для репозиторію та являє собою контейнер для зберігання та створення репозиторіїв.

2.3.1 Repository

2.3.1.1 Загальний вигляд та опис шаблону

Виходячи з рисунку загальної архітектури шаблону Repository наведеного у розділі «Аналіз предметної області» слідує, що шаблон "Repository", складається з узагальненої абстракції і більш деталізованих її імплементацій. Необхідно зазначити ща базові реалізації містять узагальнений тип, при імплементації даний тип буде підставлено в наслідок передач ічітко визначеного, конкретного типу сутності. Розглянемо приклад шаблону з точки зору реалізації в коді.

- IRepository<T> шаблоний інтерфейс що описує базову поведінку для класів наслідників по роботі з даними;
- IEntityRepository більш конкретна абстракція задачею якої є зберігання в собі унікальних методів для роботи з поточною сутністю(Entity) та наслідується від IRepository<T> закладуючи абстракцію для базових методів але з чітко визначеним типом;
- BaseRepository<T> є класом що імплементує IRepository<T> та реалізовує всі його методи базові для всіх сутностей і містить привязку до певної конкретно вибраної технології по роботі з БД;
- EntityRepository конкретний клас наслідни що імплементує свій конкретний інтерфейс (IEntityRepository) та унаслідує базовий клас BaseRepository<T> для реалізації стандартних методів по роботі з даними.

2.3.1.2 Автоматична генерація патерну як шаблону

Виходячи з опису роботи шаблону можна сказати наступне. Маємо ключовий інтерфейс, що реалізовує CRUD (Create-Read-Updtae-Delete) операції. Далі згідно з переліком сутностей генеруємо інтерфейси під кожну з них наслідуючи базову поведінку, після чого є клас що реалізовує данну поведінку описаний нами відповідно до типу сховища далі реалізуємо окремі класи репозиторіїв, що наслідують абстрактно описані методи в базовому

інтерфейсі та клас з їх реалізацією. Відповідно за необхідністю можна генерувати окремі методи з іншою логікою і не стандартною поведінкою, но дане рішення відноситься до програмної реалізації та буде описане там.

2.3.2 Unit Of Work

2.3.2.1 Загальний вигляд та опис шаблону

Виходячи з рисунку загальної архітектури шаблону Repository наведеного у розділі «Аналіз предметної області» слідує, що шаблон " Unit Of Work", є класом обгорткою для репозиторію та являє собою контейнер для зберігання та створення репозиторіїв.

Дана надбудова над репозиторієм застосовується в випадку коли в додатку відсутня DI (Dependency Injection) як механізм підстановки імплементацій за інтерфейсом.

Говорячи мовою псеевдокоду даний клс буде мати наступний вигляд :

```
public class UnitOfWork : IDisposable
{
   private OrderContext db = new OrderContext();
// В випадку використання ORM
   private IEntity1Repository entity1Repository;
   private IEntity2Repository entity2Repository;
//…
   private IEntityNRepository entityNRepository;
}
```

Як можна побачити з опису, паттерн має визначену архітектуру і є розширюваним за рахунок засобів додавання нових класів і інтерфейсів для роботи з кожною сутністю.

2.3.2.2 Автоматична генерація патерну як шаблону

Виходячи з опису роботи шаблону можна сказати наступне. Клас обгортку для агрегації всіх репозиторіїв. Відповідно задача генерації коду зводиться до генерації репозиторіїв згідно з описаним вище шаблоном, після чого

необхідно створити інтерфейс та клас що міститиме інерфейсні посилання на всі репозиторії, що в свою чергу реалізовані під всі сутності.

2.3.3 Технологічні рішення по роботі с базами даних

Однією із задач, що необхідно вирішити при реалізації програмного продукту, що працює з сховищем даних, є побудова механізму програмної роботи з даним сховищем. Для вирішення поточної задачі є два підходи:

- Qwery builders;
- ORM (Object-Relational Mapping).

Два вище названі підходи містять ряд спільних рис, таких як підключення до бази даних через connection string(строчку підключення), що зберігається в конфігураційному файлі додатку та містить необхідні дані для встановлення зєднання, а саме імя бази даних адреса серверу управління реляційної(в нашому випадку) базою даних та авторизаційні дані. Головною задачею є реалізація класів для зберігання та обробки даних із сховища.

2.3.3.1 Qwery builders

Даний спосіб програмної взаємодії з джерелом даних, що базується на передачі в базу даних відповідного блоку коду, що реалізований згідно з лексичних та семантичних стандартів мови, що застосовується на тому чи іншому сервері баз даних. Запит будується мовою T-SQL. Після передачі на виконання програмний продукт отримує відповідь про успушність чи не успішність виконання запиту і відповідний результат. Важливим фактором роботи даного способу є те, що передача запиту та сам механізм встановлення з’єднання реалізовані службовими класами технології через взаємодію з драйвером СУРБД. Технологія що вирішує дане рішення в межах платформи .NET є ADO.Net. Недоліком даного підходу є відсутність автоматично встановленої відповідності між полями класу моделі та відповідною таблицею чи результатом агрегаційної вибірки. Тобто по факту повернення результату необхідно кожному окремому прочитаному полю вручну встановити

відповідність з полем класу беручи на себе відповідальність за відповідність типів назв і тд.

2.3.3.2 Автоматизація роботи з БД з використанням Qwery builders

Повертаючись до завдання, яке повинна вирішувати технологія по програмній взаємодіїї з джерелом даних в межах шаблону Repository, а саме реалізації CRUD операцій, постає питання в практичній реалізації даного підходу. Відповідно до опису наведеного в 2.3.1.1 результатом спроби побудови рішення на основі Qwery builders виникає проблема необхідності створення та імплементації всвіх методів в кожному окремо взятому репозитарії, що в свою чергу протирічить принципам його застосування. Частковим рішенням даної проблеми буде підстановка в тіло запиту назви таблиці. Дане рішення є частковим через неможливість його застосування до більш складних вібірок.

2.3.3.3 ORM

Object-Relational Mapping – технологія що реалізує побудову зв’язку програмної моделі з відповідними таблицми, а точніше кажучи результатами вибірок. Даний зв’язок здійснюється за допомогю побудови додаткового шару технологією що надає платформа .NET. Принцип роботи даної технології полягає в побудові Proxy патерну для рішення поставленої задачі. Тобто платформа завдяки фреймворку створює набір метаданих, що зберігають відповідніть таблиці – класу, поля - рядку таблиці, і відповідність типів даних. Більшість фреймворків такого типу містять рівень кешу, що зберігає інформацію про запити, та відповідні колекції, що надають можливість відстежувати зміни в отриманих об’єктах помічаючи їх стан. Такий підхід забезпечує транзакційність запиту, оскільки реальна зміна в БД відбудеться тільки по факту виклику методу збереження змін.

В платформі .NET дану технологію реалізовує Entity Framework. Розглянемо процес створення даного фреймворку. Даний фреймворк в початкових версіях містив виключно обгортку для LINQ TO SQL. Наступними кроками розвитку даного фреймворку стала генерація бази даних

з відповідно заданої обєктно-орієнтованої моделі, заданої відповідно в вигляді класів що описують всі таблиці та в випадку необхідності автоматичного прив’язки даних до результатів часткових вибірок, або ж вибірок з композиції таблиць – класи що описують кожну з необхідних моделей. Розглядаючи механізм відстеження змін та кешування результатів необхідно розглянути службовий клас контексту бази даних. Створення власного класу контексту з переліком відповідних колекцій, що інкапсулють в собі логіку збереження станів та кешування по кожній окремій таблиці, необхідно для процесу генерації бази даних. Базовий клас контексту від якого повинен бути пронаслідуваний конкретний клас контексту користувацької бази даних, містить загальні методи по відслідковуванню змін по кожній колекції та компонує логіку поведінки всіх колекцій, що описані в ньому.Іншим підходом є створення класу контексту по існуючій базі даних.Розглянемо назви даних способів генерації і їх логіку детальніше:

– CodeFirst;
– DataBaseFirst.

Entity Framework CodeFirst

Як було зазначено в описі загальної поведінки ORM фреймворків та відповідного конкретного прикладу на якому продовжується більш детальний розгляд кожного з підходів, можна сказати що по факту опису всіх сутностей моделей, вказання необхідних атрибутів для установки тих чи інших додаткових функціональних одиниць бази даних, таких як зовнішні ключі, унікальний ідентифікатор запису і тд, опису класу контексту буде згенеровано відповідну базу даних згідно до вимог. Також існують методи, які можна перевизначити для додавання власної логіки в момент створення бази даних, а також метод, що в випадку першоразового створення надасть можливість заповнити базу даних значеннями. Після створення бази даних постає питання підтримки консистентності програмної моделі і моделі джерела даних. Для вирішення даної задачі інують міграції. В випадку зміни програмної моделі достатньо виконати команду add-migration і Orm

спывставивши метадані збережені в службову таблицю з поточними доповнить базу відповідними змінами.

Приклад та спосіб роботи міграцій зображений на Рисунку 2.7.

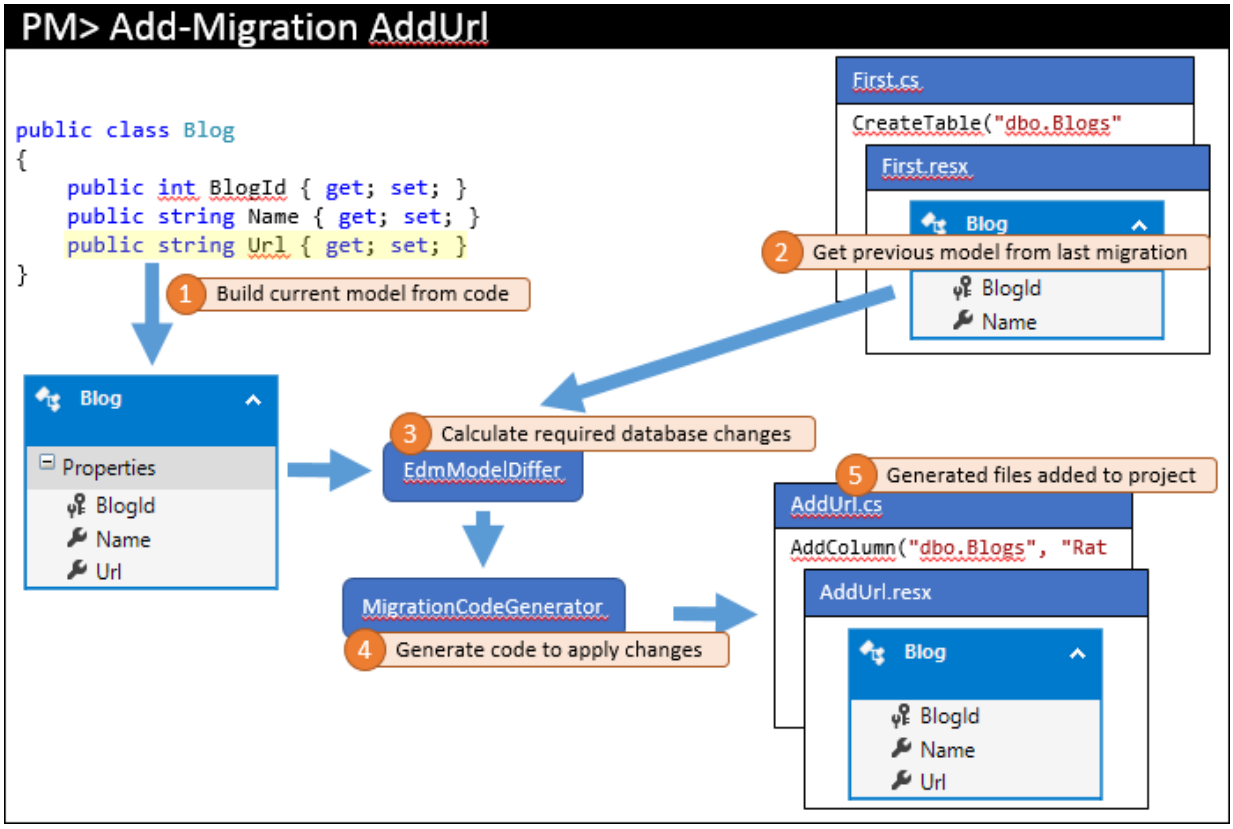

Рисунок 2.7 – Приклад моделі роботи міграції CodeFirst

Основною причиною для створення подібного рішення є складність підтримання консистентності моделей та необхідніть запуску програмістами бази даних разом з проектом в максимально короткий час.

Entity Framework DataBaseFirst

Даний підхід полягає в генерації моделей та контексту з існуючої бази даних. Недоліками даного підходу є наявність одностороннього зв’язку що вимагає внесення змін виключно в базу даних та оновлення контексту, що в свою чергу потребує передачі SQL скіптів, затрачання часу на перевірку результатів оновлення.

2.3.3.4. Автоматизація роботи з БД з використанням ORM

Виходячи з опису репозиторіїв виникає ключове питання, яким чином реалізувати CRUD операції. В випадку ORM необхідно описати Proxy обєкти, що відповідатимуть за програмну взаємодію з джерелом даних,

описати файл контексту та вказати звязки між таблицями, згідно до вимог конкретної ORM, після чого будувати логіку роботи шаблонів по роботі з дерелом даних відштовхуючись від колекції проксі обєктів через які і відбуватиметься подальша робота.

2.4 Бізнес-логіка

Виходячи з аналізу предметної області бізнес-логіка переважно полягає у обробленні даних, отриманих з рівня доступу до даних, шляхом використання поведінкових шаблонів. таких як стратегія, також класів поведінки, які описують логіку оброблення даних, відхиляючись від канонічних шаблонів, в певних випадках команд. Кожний складовий компонент реалізує вже кінцеву функціональність, яку очікує отримати користувач. Перехід між сутністю, що описує таблицю бази даних і сутністю, що виходить в ході опрацювання бізнес-логікою є посередником між рівнями області визначення і розподілу даних (domain and data mapping layers) і відповідно приводить до створення додаткового шару Data Mapper.

Згідно закладеної концепції складові компоненти є розширюваним за рахунок засобів додавання конкретних класів з вузько націленою поведінкою.

2.4.1 Паттерн стратегії

2.4.1.1 Загальний опис паттерну стратегії

Виходячи з рисунку загальної архітектури шаблону стратегії наведеного у розділі «Аналіз предметної області» шаблон стратегія складається з базового опису з відповідним методом, який приймає на вхід необхідні параметри, і відповідно кілька реалізацій даного методу.

Загальний вигляд шаблону в програмній реалізації матиме наступний вигляд.

```
public interface IStrategy
{
   void Algorithm();
}
```

```
public class ConcreteStrategy1 : IStrategy
{
    public void Algorithm()
    {}
}

public class ConcreteStrategy2 : IStrategy
{
    public void Algorithm()
    {}
}

public class Context
{
    public IStrategy ContextStrategy { get; set; }

    public Context(IStrategy _strategy)
    {
        ContextStrategy = _strategy;
    }

    public void ExecuteAlgorithm()
    {
        ContextStrategy.Algorithm();
    }
}
```

2.4.1.2 Автоматизація генерації паттерну стратегії

Виходячи з загального опису видно, що задопомогою взаємодії з користувачем визначається прототип методу поведінки в стратегії, далі

задається набір логічних поведінок що ляже в основу конкретних реалізцій стратегій поведінки, після чого програма генерує необхідний базовий інтерфейс клас контексту та відповідні конкретні реалізації.

2.4.2 Паттерн команди

2.4.2.1 Загальний опис патерну команди

Виходячи з рисунку загальної архітектури шаблону команди наведеного у розділі «Аналіз предметної області» сфера застосування патерну команди полягає у здійсненні (виконанні) тієї чи іншої логічної поведінки на вимогу, дозволяючи абстрагуватися від конкретної логіки самої команди всередині механізму.

Розглянемо приклад реалізації даного патерну.

```
abstract class Command
{
   public abstract void Execute();
   public abstract void Undo();
}
// конкретна команда
class ConcreteCommand : Command
{
   Receiver receiver;
   public ConcreteCommand(Receiver r)
   {
      receiver = r;
   }
   public override void Execute()
   {
      receiver.Operation();
   }
```

```
    public override void Undo()
    {}
}

// отримувач команди
class Receiver
{
    public void Operation()
    { }
}
// ініціатор команди
class Invoker
{
    Command command;
    public void SetCommand(Command c)
    {
        command = c;
    }
    public void Run()
    {
        command.Execute();
    }
    public void Cancel()
    {
        command.Undo();
    }
}
class Client
{
    void Main()
```

```
    {
        Invoker invoker = new Invoker();
        Receiver receiver = new Receiver();
        ConcreteCommand command=new ConcreteCommand(receiver);
        invoker.SetCommand(command);
        invoker.Run();
    }
}
```

2.4.2.2 Автоматизація генерації паттерну команди

Виходячи з загального опису видно, що задопомогою взаємодії з користувачем визначається прототип методу поведінки в відповідної стратегії, далі задається набір логічних поведінок, що ляже в основу конкретних реалізцій команд, після чого програма генерує необхідний базовий інтерфейс клас команди та відповідні конкретні реалізації обробки данної команди логіку якої встановлює користувач.

2.4.3 Паттерн ланцюг обов’язків

2.4.3.1 Загальний опис патерну ланцюг обов’язків

Виходячи з рисунку загальної архітектури шаблону ланцюг обов’язків наведеного у розділі «Аналіз предметної області» та загального опису можемо стверджувати, що данний патерн полягає в композиції викликів заздалегідь визначених обробників в конкретно заданій послідовності для кожної події та компоненту. Розглянемо реалізацію даного шаблону.

```
class Client
{
   void Main()
   {
      Handler h1 = new ConcreteHandler1();
      Handler h2 = new ConcreteHandler2();
```

```
        h1.Successor = h2;
        h1.HandleRequest(2);
    }
}
abstract class Handler
{
    public Handler Successor { get; set; }
    public abstract void HandleRequest(int condition);
}

class ConcreteHandler1 : Handler
{
    public override void HandleRequest(int condition)
    {
        if (condition == 1)
        {
            // обработка;
        }
        else if (Successor != null)
        {
            Successor.HandleRequest(condition);
        }
    }
}

class ConcreteHandler2 : Handler
{
    public override void HandleRequest(int condition)
    {
        if (condition==2)
```

```
        {
            // обробка;
        }
        else if (Successor != null)
        {
            Successor.HandleRequest(condition);
        }
    }
}
```

Оброблювач визначає загальний для всіх конкретних оброблювачів інтерфейс. Зазвичай достатньо описати єдиний метод обробки запитів, але може мати місце оголошення і метод виставлення наступного обробника. Тобто в ході делегування методу до кінцевої точки за рахунок обходу ланцюжку викликів і передачі відповідальності за рішення до конкретного компоненту якій здатен правильно опрацювати даний запит та на якому припиняється подальший обхід ланцюга .

2.4.3.2 Автоматизація генерації паттерну ланцюг обов’язків

Виходячи з загального опису видно, що задопомогою взаємодії з користувачем визначається прототип методу поведінки кожного окремо визначеного обробника, після чого програма генерує необхідний базовий інтерфейс та клас обробника. Наступним кроком є опис ланцюгувиклику для тієї чи іншої заданої задачі.

2.4.4 Паттерн посередник

2.4.4.1 Загальний опис патерну посередник

Виходячи з рисунку загальної архітектури шаблону посередник наведеного у розділі «Аналіз предметної області» та загального опису, можемо стверджувати, що даний патерн визначає інтерфейс для обміну інформацією з компонентами. Зазвичай вистачає одного методу, щоб повідомляти

посередника про події, що сталися в компонентах. В параметрах цього методу можна передавати деталі події: посилання на компонент, в якому вони є і будь-які інші дані.

Розглянемо приклад реалізації даного патерну.

```
abstract class Mediator
{
   public abstract void Send(string msg, Colleague colleague);
}

abstract class Colleague
{
   protected Mediator mediator;

   public Colleague(Mediator mediator)
   {
      this.mediator = mediator;
   }
}

class ConcreteColleague1 : Colleague
{
   public ConcreteColleague1(Mediator mediator)
      : base(mediator)
   { }

   public void Send(string message)
   {
      mediator.Send(message, this);
   }
```

```
    public void Notify(string message)
    { }
}

class ConcreteColleague2 : Colleague
{
    public ConcreteColleague2(Mediator mediator)
        : base(mediator)
    { }

    public void Send(string message)
    {
        mediator.Send(message, this);
    }

    public void Notify(string message)
    { }
}

class ConcreteMediator : Mediator
{
    public ConcreteColleague1 Colleague1 { get; set; }
    public ConcreteColleague2 Colleague2 { get; set; }
    public override void Send(string msg, Colleague colleague)
    {
        if (Colleague1 == colleague)
            Colleague2.Notify(msg);
        else
            Colleague1.Notify(msg);
    }
```

```
}
```

Даний паттерн вирішує задачу абстрагування від жорстких зав’язків та довгих ланцюгів виклику.

2.4.4.2 Автоматизація генерації паттерну ланцюг обов’язків

Виходячи з загального опису видно, що задопомогою взаємодії з користувачем обира.ться необхідні компоненти після чого створюється посередник і задаються методи нотифікацій з тим чи іншим переліком компонентів. Відповідно генерується абстрація в вигляді інтерфейсу та конкретика в вигляді класу що описує і реалізовує поведінку нотифікацій.

2.4.5 Паттерн стан

2.4.5.1 Загальний опис патерну стан

Виходячи з рисунку загальної архітектури шаблону посередник наведеного у розділі «Аналіз предметної області» та загального опису даний шаблон пропонує створити окремі класи для кожного стану, в якому може перебувати об'єкт, а потім винести туди поведінки, які відповідають цим станам замість того, щоб зберігати код всіх станів.

Розглянемо приклад реалізації даного шаблону.

```
class Program
{
   static void Main()
   {
      Context context = new Context(new StateA());
      context.Request(); // Переход в состояние StateB
      context.Request();  // Переход в состояние StateA
   }
}
abstract class State
{
```

```
    public abstract void Handle(Context context);
}
class StateA : State
{
    public override void Handle(Context context)
    {
        context.State = new StateB();
    }
}
class StateB : State
{
    public override void Handle(Context context)
    {
        context.State = new StateA();
    }
}

class Context
{
    public State State { get; set; }
    public Context(State state)
    {
        this.State = state;
    }
    public void Request()
    {
        this.State.Handle(this);
    }
}
```

2.4.5.2 Автоматизація генерації паттерну стратегії

Виходячи з загального опису видно, що задопомогою взаємодії з користувачем визначається прототип методів поведінки що необхідні в кожному конкретному стані, після чого задається контекст і логіка переходів станів, після чого програма генерує необхідний базовий інтерфейс клас контексту та відповідні конкретні реалізації станів.

2.5 Представлення

2.5.1 Загальний опис представлення

Виходячи з аналізу предметної області, маємо:

- Для реалізації зовняшньої точки доступу до додатку використовуються шаблони, що розмежовують представлення від логіки.
- Найчастіше вживаним шаблоном є MVC.
- Моделі для відображення отримуються внаслідок виклику методів бізнес-логіки

Розглянемо структурну особливість шаблону та його компоненти:

- Моделі. Об'єкти моделей є частинами програми, що реалізують логіку для домену даних програми. Об'єкти моделей часто отримують і зберігають стан моделі в базі даних. У невеликих додатках ця модель має на увазі концептуальний, а не фізичний поділ. Наприклад, якщо програма тільки зчитує набір даних і відправляє його в уявлення, то фізичний шар моделі і пов'язаних класів відсутня. У цьому випадку набір даних приймає роль об'єкта моделі.
- Подання. Уявлення служать для відображення інтерфейсу програми. Інтерфейс користувача зазвичай створюється на основі даних моделі.

- Контролери. Контролери здійснюють взаємодію з користувачем, роботу з моделлю, а також вибір подання, що відображає користувальницький інтерфейс. У додатку MVC подання лише відображають дані, а контролер обробляє дані, що вводяться і відповідає на дії користувача. Наприклад, контролер може обробляти рядкові значення запиту і передавати їх у модель, яка може використовувати ці значення для відправки запиту до бази даних.

Класифікація за моделями MVC додатків:

1. З пасивної моделлю - модель не має жодних способів впливати на уявлення або контролер, і використовується ними в якості джерела даних для відображення. Всі зміни моделі відслідковуються контролером і він же відповідає за перемальовування подання, якщо це необхідно. Така модель частіше використовується в структурному програмуванні, так як в цьому випадку модель являє просто структуру даних, без методів їх обробних.

2. З активною моделлю - модель оповіщає уявлення про те, що в ній відбулися зміни, а уявлення, які зацікавлені в оповіщенні, підписуються на ці повідомлення. Це дозволяє зберегти незалежність моделі, як від контролера, так і від уявлення.

Виходячи з детального опису патерну можна говорити про те, що контролер виступає з’єднувачем безпосередньо бізнес-логіки і представлення. В рамках шаблонізації веб додатків відзначимо, що контроллер повинен містити в собі весь перелік методів по роботі з однією сутністю простою чи компонованою, що говорить про те що повинен бути згенерований CRUD операцій для кожного елементу даних моделі і відповідно створене представлення під кожен метод.

2.5.2 Автоматизація генерації представлень

Відповідно до наданого вище опису шаблонізацію створення коду можна розділити. За структурною характеристикою:

– контролери;
– представлення;

За функціональною характеристикою:

– CRUD операції;
– поведінкові;

Для першого типу можлива повна автоматична генерація рішення на основі моделей. Тобто для кожної сутності буде згенеровано свій контролер з відповідними методами по роботі з даною сутністю.

Для генерації поведінкових функцій системи необхідно вказувати метод його сигнатуру і відповідно підключати окремі компоненти бізнес-логіки таким чином, щоб була відповідність між результатом отриманим в методі і типом значення, що буде повернено в сигнатурі методу.

2.6 Загальна модель шаблонізації веб-додатків

Відповідно до вище розглянутого можна сказати, що було розглянуто кожну складову і спосіб її реалізації та шаблонізації, розглянемо загальний вигляд даного процесу. Дотримуючись наведеного описання, задачу створення будь-якого web-застосування можна розглядати як задачу заповнення певного шаблону. Варто однак зазначити, що побудова навіть за шаблоном коректної архітектури, написання монотонного и муторного коду вимагають багато часу. Тому й постає проблема автоматизації створення інформаційної системи шляхом автоматизованої реалізації шаблона системи на підставі вимог користувача. Вирішення зазначеної проблеми логічно розпочати з конструювання моделі бази даних і генерування рівня доступу до неї. Наступним кроком є реалізація обробки даних засобами вибудуваних стратегій на вимогу користувача. Насамкінець, необхідно генерувати представлення.

Виходячи з вище викладеного, можна описати будову програми, її загальний вигляд. За модель будови програми будемо використовувати діаграму станів.

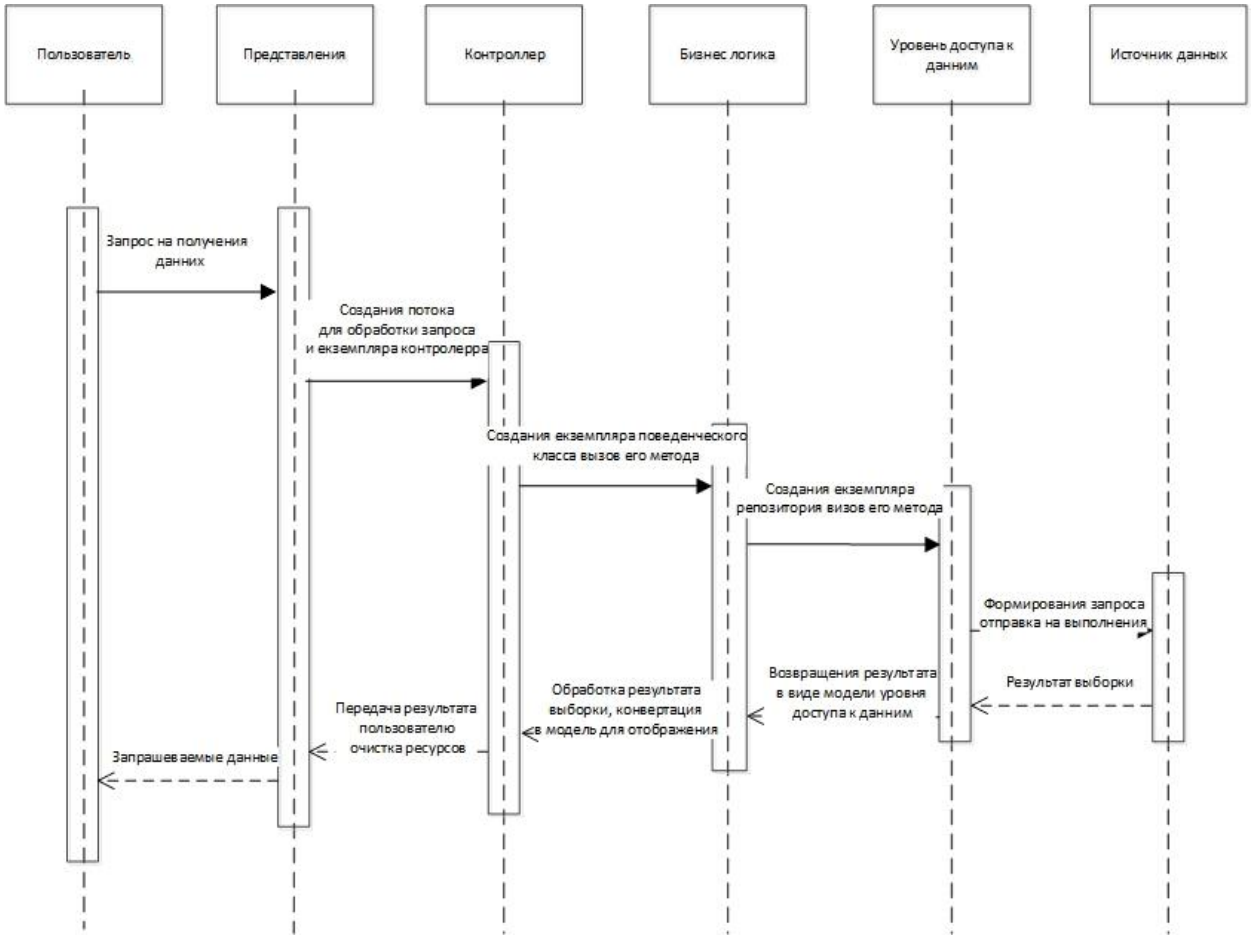

Рисунок 2.8 – загальний вигляд процесу обробки запиту користувача

Виходячи з діаграми наведеної на рисунку в випадку запиту користувача за допомогою HTTP протоколу запит надсилається на сервер після чого створюється потік та відповідно стек потоку в якому зберігаються всі об’єкти необхідні для обробки поточного запиту. Після чого запит направляється відповідному контролеру, в момент направлення технологія створює безпосередньо сам контролер реєструє всі його залежності і створює цілий граф об’єктів до самого низького рівня. Далі виконується код методу в контролері що викликає методи бізнес-логіки далі доступ до джерела інформації і відповідно побудова результату і його повернення користувачеві.

2.7 Висновки

В даному розділі було розглянуто деталізовано трирівневу архітектуру, визначено комплексний підхід до питання шаблонізації коду та його автогенерації. Конкретизовано загальний огляд шаблонів наведено програмно

реалізацію та виділено підходи до автоматизації в кожному окремо взятому шаблоні згідно з принципу декомпозиції елементів сучасної web-системи.

3 МАТЕМАТИЧНА МОДЕЛЬ

Видається доцільним для вибору і інтеграції компонентів в комплесне рішення використовувати каузальну логіку першого порядку, структурні елементи якої описані авторами для інтеграції додатків в праці [1].

Символи:

службові: (,), [,], {, }, :, <, >, ,;

константи:

1) *індивідні* основних типів (int, real, char, bool) - a_1^1, a_2^1, …, a_1^2, a_2^2, ... де кожна константа a_i^k має тип (основний тип) *k;* структурного типу (конструкти) - c_1, c_2, ...; структурного типу - d_1, d_2, ...; обєктного типу (проблема, сутність, відношення) — e_1, e_2, ...;

2) *функціональні і-місні,* для індивідів типу *k* - h_1^1, h_2^1, …, h_1^2, h_2^2, ...;

3) *предикатні і-місні,* для індивідів типу для індивідів типу *k* - A_1^1, A_2^1, …, A_1^2, A_2^2, ... (цей клас включає таксономічні, реляційні й інші предикати, а також традиційні відношення, щонайменше рівності = і порядку ≥);

змінні: для індивідів типу k - x_1^1, x_2^1, …, x_1^2, x_2^2, ..., де кожна змінна x_i^k належить типу *k;*

логічні: $\neg, \wedge, \vee, \leftarrow, \exists, \forall, \Leftrightarrow$.

Індивідні терми типу *k:*

1) кожна індивідна константа a_i^k типу *k* є індивідним термом типу *k;*

2) кожна не зв'язана змінна x_i^k для індивідів типу *k* є індивідним термом типу *k*;

3) якщо h_i^j є певною функціональною константою для індивідів типу *k* і τ_1, …, τ_j є термами для індивідів типу *k*, то h_i^j (τ_1, …, τ_j) є індивідним термом типу *k;*

4) немає інших індивідних термів типу *k*.

Терми, отримані за допомогою правил 1 або 2 цього визначення називатимемо простими, а всі інші — складними.

Формули для індивідів:

1) якщо A_i^j є предикатною константою для індивідів і $\tau_1, \ldots, \tau_j$ є термами для індивідів, то $A_i^j(\tau_1, \ldots, \tau_j)$ є атомарною формулою для індивідів;

2) атомарна формула для індивідів є формулою для індивідів;

3) немає інших формул для індивідів.

Тут і надалі розглядаємо лише системи клауз Хорна (з єдиним символом → і атомарними формулами ліворуч і праворуч від нього, неявним квантором $\forall$).

Специфікатори конструкцій є конструкціями типу τ_1, де τ_1 є термом для індивідних об'єктів. Специфікаторами конструкцій є**:**

1) якщо $e_1^e \ldots e_i^e$ є індивідними термами типу сутності, і $e_1^r \ldots e_i^r$ є індивідними термами типу відношення, і $A_1^j(a_1^1 \ldots a_j^1) \ldots A_i^j(a_1^k \ldots a_j^k)$ є атомарними формулами для індивідів основних типів, і τ є індивідним термом типу конструкції, то τ: $(e_1^e \ldots e_i^e, e_1^r \ldots e_i^r, A_1^j(a_1^1 \ldots a_j^1) \ldots A_i^j(a_1^k \ldots a_j^k))$ специфікаторами конструкцій.

2) немає інших специфікаторів конструкцій.

Передумови:

1) якщо c_1 є специфікатором конструкції, і П є послідовністю атомарних формул, то $<\tau_1:(c_1,П)>$ є елементарною передумовою;

2) елементарна передумова є передумовою;

3) якщо $<\tau_1>$ є передумовою, і τ_2 є елементарною передумовою, то $<\tau_1, \tau_2>$ є передумовою;

4) немає інших передумов.

Постумови:

1) якщо τ_1 є специфікатором конструкції, і П послідовністю атомарних формул, то $<\tau_1:(c_1,П)>$ є елементарною постумовою;

2) елементарна постумова є постумовою;

3) якщо $<\tau_1>$ є постумовою, і τ_2 є елементарною постумовою, то $<\tau_1, \tau_2>$ є постумовою;

4) немає інших постумов.

Специфікатори методів:

1) якщо τ є індивідним термом типу метод, і $<\tau_1>$ є передумовою, і $< \tau_2>$ є постумовою, то τ: ($<\tau_1>$, $< \tau_2>$) є специфікатором методу;

2) арність по конструкціям передумови методу не має бути меншою від арності по конструкціям постумови методу;

3) немає інших специфікаторів методів.

Специфікатори проблем:

1) якщо $<\tau_1>$ є передумовою, $<\tau_2>$ є постумовою і τ є передумовою для індивідного об'єкту типу Problem, то τ: $<< \tau_1>,< \tau_2>>$ є специфікатором проблеми;

2) немає інших специфікаторів проблем.

Клаузою є вираз вигляду $\Pi \rightarrow \Lambda$, де Π є послідовністю атомарних формул; Λ є єдиною атомарною формулою. Клаузи поділяються на індивідні (містять лише атомарні формули для індивідів), типізовані (містять лише атомарні формули для типів) і загальні (містять атомарні формули для індивідів і типів).

База знань системи складається з трьох основних частин. У першій частині онтологічні аксіоми описують методи і інші компоненти доступних для використання фреймворків. У другій частині подається онтологія системи, описана за допомогою мов OWL базованих на RDF. У другій частині зведені правила виведення, необхідні для отримання бажаного для користувача результату. Тут є правила виведення двох типів. Правила виведення першого типу використовуються для інтеграції багатьох методів та інших компонентів у єдиний додаток. Вони враховують особливості проблеми, передумови, постумови, описи методів та інших компонентів системи. Реалізоване з

бажаним результатом виведення певним чином визначає архітектуру додатку, який створюється, в рамках визначеного класу архітектур.

Правила виведення першого типу:

1. Якщо d_1: (<τ_1>,< τ_2>) і d_2: (<τ_3>,< τ_1>) то d_1: (<d_2>,< τ_2>)

2. Якщо d_1: (<τ_1>,< τ_2>) і d_2: (<$\tau_3 \wedge \tau_4$>,< τ_1>) то d_1: (<d_2>,< τ_2>)

3. Якщо d_1: (<$\tau_1 \wedge \tau_2$>,< τ_3>) і d_2: (<τ_4>,< τ_1>) і d_3: (<τ_5>,< τ_2>) то d_1: (<$d_2 \wedge d_3$>,< τ_3 >)

4. Якщо d_1: (<τ_1>,< τ_2>) і d_2: (<$\tau_3 \vee \tau_4$ >,< τ_1>) то d_1: (<d_2>,< τ_2>)

5. Якщо d_1: (<$\tau_2 \wedge \tau_2$>,< τ_1>) то d_1: (<τ_2>,< τ_1>)

6. Якщо d_1: (<τ_1, τ_2>,< τ_3>) і d_2: (<τ_4>,< τ_1>) і d_3: (<τ_5>,< τ_2>) то d_1: (<d_2, d_3>,< τ_2>)

7. Якщо d_1: (<τ_1>,< τ_2, τ_3>) і d_2: (<τ_2>,< τ_4>) і d_3: (<τ_3>,< τ_5>) то d_2: (<d_1>,< τ_4>) і d_3: (<d_1>,< τ_5>)

Правила виведення другого типу використовуються для пришвидшення процесу виведення. Вони враховують семантику проблеми користувача і семантику методів та інших компонентів системи, виражену в термінах онтології системи. По суті, ці правила використовуються для скорочення перебору правил і аксіом першої частини бази знань системи. Вони описують допустимі семантично на рівні вхідних і вихідних сутностей варіанти комбінування методів та інших компонентів системи у більш функціонально повні їх комбінації. Ці комбінації можна використовувати для виведення у просторі правил першого типу. Змістовно використання правил виведення другого типу можна ілюструвати процесом 3D-візуалізації поєднання методів та інших компонентів системи у просторові конструкції на основі спільних сутностей на їх входах/виходах.

5 Механізм виведення керований семантикою. Для виведення будемо використовувати підхід, запропонований авторами в праці [1]. Мова йде власне про процедури виведення і відновлення дерева виведення. Але доповнимо цей підхід попередньою процедурою пошуку просторової

конструкції пов’язаних за входами/виходами методів та інших компонентів системи, яка на входах має сутності визначені користувачем як вхідна інформація, а на її виходах маємо сутності, визначені користувачем як вихідна інформація. Визначимо низку понять, необхідних для подання процедур виведення.

Природним чином орієнуємося на засоби представлення знань і для скорочення перебору використовуємо базу знань, насамперед накопичену інформацію щодо ефективних схем виведення для часто виконуваних запитів і її можливості структурування, факторизації і абстрагування. Мова йде про комбіновану стратегію виведення, на нижніх рівнях якої блокується породження збиткових резольвент. Формально мова йде про комбінування типізації тверждень Р. Ковальскі і методу аналогій Д. Плейстеда. При цьому виведення в абстрактному просторі, результат якого використовується потім для управління процесом доведення в початковому просторі пошуку рішень, виключно базується на аксіомах і правилах виведення бази знань.

Але відсікання значної частини безперспективних гілок виведення у початковому просторі виконуємо у два етапи. На першому етапі формуємо конструкції пов’язаних за входами/виходами методів та інших компонентів системи. На другому етапі використовуємо виведення у абстрактному просторі. За відображення переходу к до абстрактного простору будемо використовувати таксономічні зв’язки. Тоді виведення в абстрактном просторі зведеться до виведення в системі типів (класів сутностей) з поступовим заглибленням в систему підтипів аж до індивідів. Як і раніше для скорочення перебору як в абстрактному (для класів і типів), так і в початковому (для індивідів) просторах використовуватимемо модифікації стратегії резолюції Робінсона. Потрібні для взаємної адаптації методу аналогій і модифікацій стратегії резолюції властивості використовувані відображення набувають природним чином на відповілно структурованій базі знань.

Для методу аналогій використовуємо поняття мультиклаузи (*m*-клаузи) як мультимножини атомарних формул (при цьому в *m*-клаузі атомарна формула *L* записується стільки раз, скільки раз вона повторюється. Звичайна клауза є *m*-клаузою з неповторюваними атомарними формулами. Над *m*-клаузами природним чином виконуються операції ∪ (обєднання), ∩ (перетин), — (різниця), · (конкатенація) і визначається відношення ⊆ (входження) для мультимножин (операції виконуються для лівої і правої частин клаузи окремо).

Нехай $A_1 \in C_1$, $A_2 \in C_2$, а α_1 і α_2 є підстановками, які дозволяють отримати найбільший спільний уніфікатор для атомарних формул A_1, A_2. Тоді клауза отримана об'єднанням $C_1\alpha_1$ та $C_2\alpha_2$ і вилученням *L* зправа і зліва від символу →, називається *m*-резольвентою *m*-клауз C_1 і C_2. Якщо ж клаузи C_1, C_2 впорядковані і *m*-резольвента отримується вилученням непідкресленої атомарної формули у обох частинах, причому у лівій частині за нею немає іншої атомарної формули, то отримуємо упорядковану лінійну *m*-резольвенту. Якщо остання атомарна формула ліворуч від символу → упорядкованої *m*-клаузи уніфікується з підкресленою атомарною формулою праворуч від символу → цієї ж клаузи, то упорядкована лінійна *m*-резольвента отримується редукцією *m*-клаузи.

Визначення *m*-клаузи і *m*-резольвенти використаємо для визначення упорядкованого лінійного m-резолюційного виводу. Розпочнемо з визначення *m*-резолюційного виводу *Tm* як пари <*V*,*Th*>, де *V* — множина вершин виводу, *Th* — множина трійок вершин. Надалі перший і другий компоненти *Tm* будемо виділяти за допомогою функцій-селекторів *s*-*N*(*Tm*) і *s*-*M*(*Tm*) відповідно. Кожна вершина $n \in$ *s*-*N*(*Tm*) виводу *Tm* характеризується позначкою *s*-*L*(*n*) і глибиною *s*-*D*(*n*). Якщо $<n_1, n_2, n_3> \in$ *s*-*M*(*Tm*), то *s*-*L*(n_3) є m-резольвентою *s*-*L*(n_1) і *s*-*L*(n_2) і кожна трійка такого вигляду називається *m*-резолюцією. У виведенні *Tm* вершина $n \in$ *s*-*N*(*Tm*) називається початковою, якщо не є третьою складовою жодної з трійок з *s*-*M*(*Tm*) (її позначка — початкова *m*-клауза), або

термінальною, якщо не є ні першою, ні другою складовою жодної з трійок з *s-M*(*Tm*) (її позначка — термінальна *m*-клауза).

Для *Tm* завжди виконується така умова: якщо $<n_1, n_2, n_3> \in$ *s-M*(*Tm*), то $< n_2, n_1, n_3> \in$ *s-M*(*Tm*). Значення глибини (у виводі) кожної вершини $n \in$ *s-N*(*Tm*) за визначенням: 1) *s-D*(*n*) = 0 для кожної початкової вершини *n;* 2) *s-D*(*n*) = 1 + *min*{*max*{*s-D*(n_1), *s-D*(n_2)}|$<n_1, n_2, n> \in$ *s-M*(*Tm*)} для кожної непочаткової вершини *n*.

Тепер назвемо *m*-резолюційний вивід *Tm* виводом із *S*, якщо позначки початкових вершин *Tm* належать множині *m*-клауз *S*. Із *S* виводимо *C*, якщо *Tm* є виводом із *S*, а *C* — позначкою однієї з вершин *Tm*. Нехай Tm_1 і Tm_2 є *m*-резолюційнми виводами. Тоді Tm_1 є субвиводом Tm_2 (позначимо $Tm_1 \subseteq Tm_2$), якщо виконуються такі умови: $s\text{-}N(Tm_1) \subseteq s\text{-}N(Tm_2)$ і $s\text{-}M(Tm_1) \subseteq s\text{-}M(Tm_2)$. При цьому, якщо всі початкові вершини Tm_1 є початковими вершинами Tm_2, то Tm_1 називатимемо початковим субвиводом Tm_2.

Насамкінець упорядкованим лінійним m-резолюційним виводом із *S* назвемо *m*-резолюційний вивід *Tm* із *S*, всі m-клаузи якого упорядковані і для довільної трійки $<n_1, n_2, n_3>$ із $s\text{-}M(Tm)$ $s\text{-}L(n_3)$ є упорядкованою лінійною *m*-резольвентою $s\text{-}L(n_1)$ і $s\text{-}L(n_2)$. За визначенням *m*-резолюційного виводу: $max\{s\text{-}D(n_1), s\text{-}D(n_2)\} = s\text{-}D(n_3) - 1$.

Упорядковані лінійні m-резолюційні виводи успадковують властивості *m*-резолюційних виводів, маючи однак свої особливості. Так, для упорядкованого лінійного m-резолюційного виводу *Tm* виконуються такі умови: 1) якщо $< n_1, n_2, n_3> \subseteq s\text{-}M(Tm)$, то $< n_2, n_1, n_3> \notin s\text{-}M(Tm)$; 2) для кожної непочаткової вершини n_3 глибини *r* існує трійка $<n_1, n_2, n_3>$, у якій n_1 має глибину *r*-1, n_2 відповідає початковій вершині, або є відсутньою (застосована редукція); 3) *Tm* містить єдину термінальну клаузу.

Для управління упорядкованим лінійним m-резолюційним виводом використовується абстракція типізації. Нехай *f* є відображенням із множини *m*-клауз на множину підмножин *m*-клауз таким, що: 1) якщо *m*-клауза C_3 є *m*-резольвентою *m*-клауз C_1 та C_2 і $D_3 \in f(C_3)$, то існують $D_1 \in f(C_1)$ і $D_2 \in f(C_2)$

такі, що результат підстановки деякої *m*-резольвенти D_1 і D_2 належить D_3; 2) $f(\varnothing)=\{\varnothing\}$; 3) якщо результат деякої підстановки *m*-клаузи C_1 належить *m*-клаузі C_2, то для довільної абстракції D_2 для C_2 маємо абстракцію D_1 для C_1 таку, що результат підстановки D_1 належить D_2. Назвемо таке відображення *f*-*m*-абстрактним, а довільне *D* із *f(C)* - *m*-абстракцією. Під відображенням типізації будемо розуміти певне відображення ϕ із атомарних формул в атомарні формули, яке відображує кожну атомарну формулу в формулу, терми якої мають тип, найближчий по ієрархії до базових типів. Відображення типізації ϕ є *m*-відображенням.

Упорядковані лінійні m-резолюційні виводи *Tm* і *Um* знаходяться у відношенні $\rightarrow_f$ (позначимо $Tm \rightarrow_f Um$), де *f* є *m*-відображенням, якщо вершини виводів *Tm* і *Um* знаходяться у такому відношенні відповідності *R*, що: 1) є істинними твердження $\forall n(n \in s-N(Tm))\ \exists n'\ (n' \in s-N(Um))\ \ (nRn')$ і $\forall n(n \in s-N(Um))\ \exists n'\ (n' \in s-N(Tm))\ \ (nRn')$; 2) якщо nRn', де $n \in s-N(Tm)$, $n' \in s-N(Um)$, то n і n' можуть бути початковими або термінальними вершинами тільки одночасно; 3) для термінальних вершин *Tm* і *Um* відношення *R* є відношенням один до одного; 4) якщо $<n_1, n_2, n_3> \in$ *s-M*(*Tm*), $<n_1', n_2', n_3'> \in$ *s-M (Um)* і $n_3 R n_3'$, то $n_1 R n_1'$ і $n_2 R n_2'$; 5) якщо n і n' є початковими вершинами виводів відповідно *Tm* і *Um* і nRn', то *s-L*(n') ∈ *f(s-L(n))*; 6) якщо n і n' є непочатковими вершинами виводів *Tm* і *Um*, відповідно, і nRn', то приклад *s-L*(n') належить *f(s-L(n)).*

Упорядкованим лінійним m-резолюційним виводом властива мінімальність — вони мають рівно одну термінальну вершину і, якщо трійка $<n_1, n_2, n_3>$ належить мінімальному упорядкованому лінійному m-резолюційному виводу, то ніяка інша трійка не може містити вершину n_3 як третю компоненту, за винятком трійки $<n_1, n_2, n_3>$.

Побудова виводу розпочинається по формулюванню проблеми користувачем. По суті, при цьому перевіряється можливість задоволення запиту з урахуванням всіх наявних ресурсів. Маючи доступ до описів усіх

модулів та інших компонентів та аксіом, які визначають можливості їх застосування, механізм виведення комбінує модулі та інші компоненти в структуру (вивід), яка забезпечує одержання результату, починаючи з вхідних даних.

Нехай упорядкований лінійний m-резолюційний вивід отримано згідно визначення як пара множин: вершин $V = \{ k_1 , k_2 , k_3 .. k_n \}$ і трійок вершин $Th = \{< k_1 , k_2 , k_3 >, < k_3 , k_4 , k_5 > ... < k_{n-2} , k_{n-1} , k_n >\}$, де k_i , $i = 1,\ldots,$ n – вершини виводу, k_n - термінальна вершина. Вивід формується, приймаючи за початкові вершини постумову поставленої проблеми і, як бокову вершину - відповідну аксіому із бази знань. Аксіома є відповідною, якщо послідовність атомарних формул, яка відповідає цій вершині виводу, або будь-яка її частина, становлять постумову аксіоми. Третя вершина цієї трійки отримується застосуванням аксіоми до формули постумови, з урахуванням правил виведення. Третя вершина першої трійки стає першою вершиною другої трійки, і процес повторюється доти, доки не буде досягнута термінальна вершина – передумова проблеми. Якщо атомарна формула, яка відповідає третій вершині трійки може бути спрощена застосуванням одного із правил виведення, вона спрощується у наступній трійці, яка буде мати лише дві вершини – вершину зі спрощуваною формулою і вершину зі спрощеною формулою.

Пошук відповідних аксіом відбувається шляхом співставлення конструкта постумови, яка опрацьовується в даний момент, з конструктами аксіом із онтології. Таким чином, із онтології вибирається множина аксіом, які описують методи, що оброблюють необхідний нам тип і формат об’єктів. Підбір і перебір аксіом під час роботи механізму виведення здійснюється тільки на даній множині.

Якщо в процесі пошуку аксіом в базі знань для чергової вершини відповідних аксіом знаходиться декілька, то кожна з цих аксіом застосовується для подальшої побудови власної «паралельної» версії виводу. Таким чином, під час роботи механізму виведення може формуватися деяка кількість виводів

різної довжини і складності, в залежості від кількості аксіом в базі знань і комбінацій їх застосування до кожної вершини. По завершению роботи механізму виведення, із множини отриманих виводів вибирається вивід з найменшою кількістю трійок вершин і найменшою кількістю застосованих аксіом. Довші, циклічні і тупикові виводи відкидаються.

Для скорочення такого перебору система попередньо підбирає структури повязаних методів та інших компонентів, сутності на входах і виходах яких є спільними. І лише на другому етапі, коли вже структури підібрані, перевіряються передумови і післяумови і визначається по суті архітектура рішення.

Множина формул і клауз, яка відповідає вершинам трійок виведення і відображає застосування аксіом першої частини і правил виведення першого типу при його побудові, є виходом механізму виведення. Для відновлення структури використовуємо алгоритм відновлення дерева виведення, запропонований авторами у праці [1].

4 ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1 Архітектура інформаційної технології

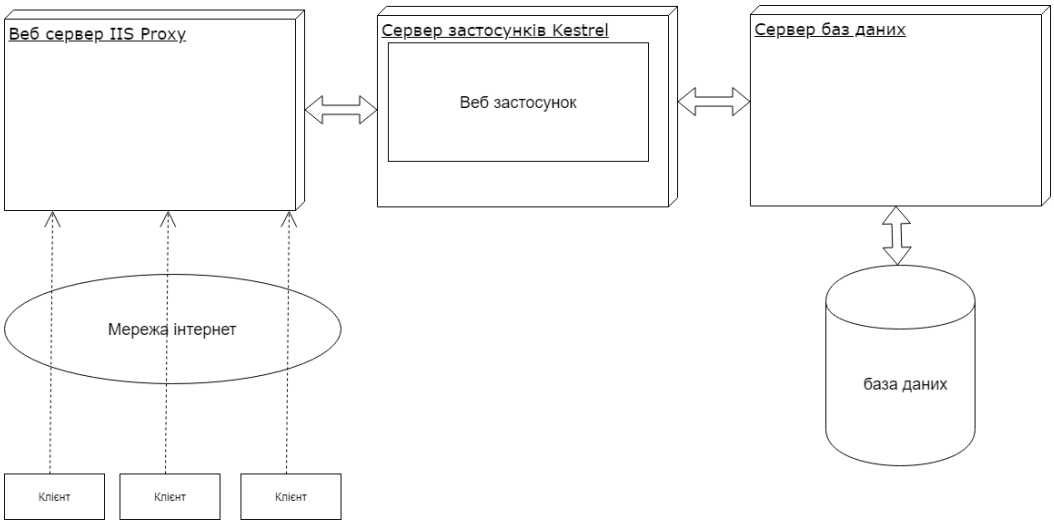

Рисунок 4.1 – архітектура інформаційної технології

На рис.4.1 представлено архітектуру інформаційної технології, котра складається з веб серверу, серверу застосунків, та серверу баз данних.

Відповідно до обраної технології в якості веб серверу використано стандартне рішення – IIS Proxy server.

В якості серверу застосувань було обрано стандартне рішення для технології Kestrel.

Аргументація вибору типу та конкретного серверу баз данних наведене в розділі 4.5. мовні аспекти.

4.1.1 Веб сервер IIS Proxy

Веб-сервер – компютерна система, що оперує данними за допомогою HTTP протоколу. Веб сервер відповідає за опрацювання вхідних запитів та повернення результату їх виконання.

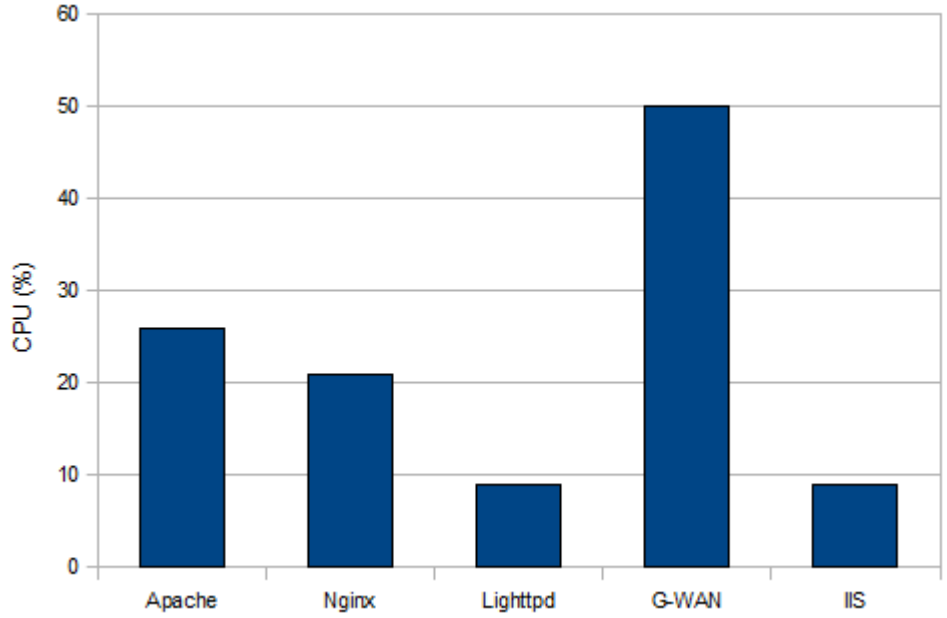

Рисунок 4.2 - навантаження ЦПУ серверу

На рис.4.2 зображено діаграму навантаженості центрального процесору веб-серверу за однакових умов(характеристик машини, кількості звернень, обєму трафіку). Як видно з рис.2 IIS веб-сервер значно випереджає конкурентів.

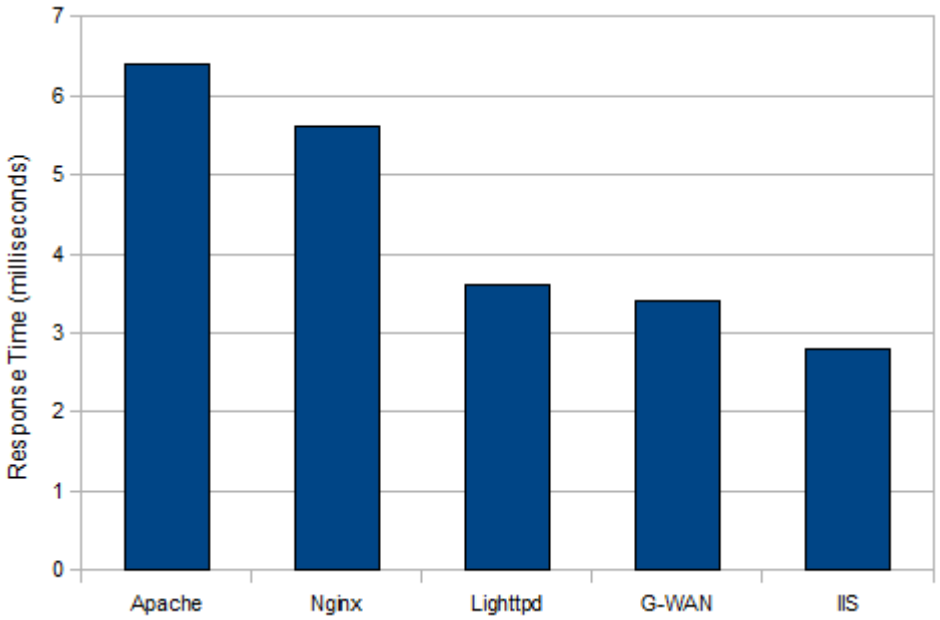

Рисунок 4.3 – швидкість відгуку

На рис.4.3 зображено діаграму швидкості відгуку веб-серверу за однакових умов(характеристик машини, кількості звернень, обєму трафіку). Як видно з рис.3 IIS веб-сервер значно випереджає конкурентів.

4.1.2 Сервер застосунків Kestrel

Kestrel представляє кросплатформенний веб-сервер, заснований на кросплатформенній бібліотеці асинхронного введення / виводу libuv. Kestrel використовує сокети і повністю складається з керованого коду. Kestrel розвивається як opensource-проект, і при необхідності на гітхабе можна подивитися його вихідний код.

Згідно з наведеною на рисунку архітектурою веб-сервер IIS відіграє роль

Proxy серверу, та перенаправляє запити до серверу застосунків Kestrel, котрий в свою чергу маршрутиризує запит між застосуваннями отрамавши результат його опрацювання повертає його Proxy серверу.

4.2 Структура інформаційної технології

4.2.1 Структура бази даних

Розглянемо рисунок 4.4 на ньому продемонстровано схему бази даних. Поточну базу даних умовно можна поділити на три логічні сутності:

- Авторизаційні дані користувачів;
- Користувацькі проекти;
- Бібліотека шаблонів;

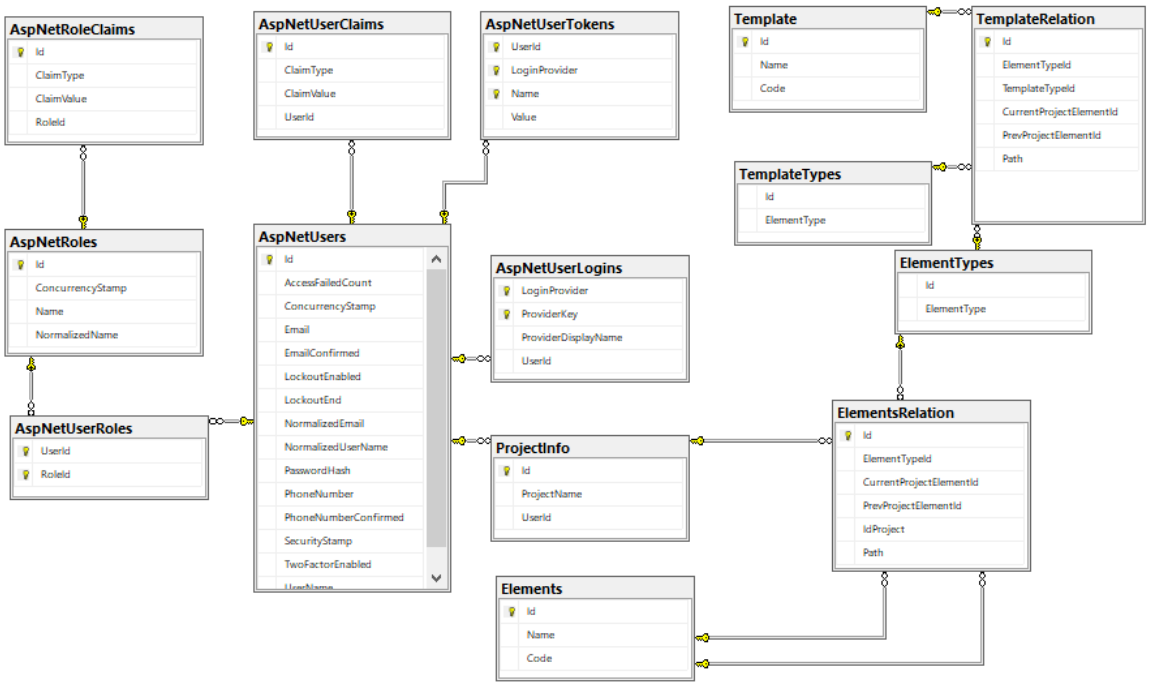

Рисунок 4.4 – схема бази даних

Відповідно таблиці представлені на рис. 4.5 - таблиці котрі необхідні механізму авторизації Identity, що вбудовано в технологію реалізації .net core mvc.

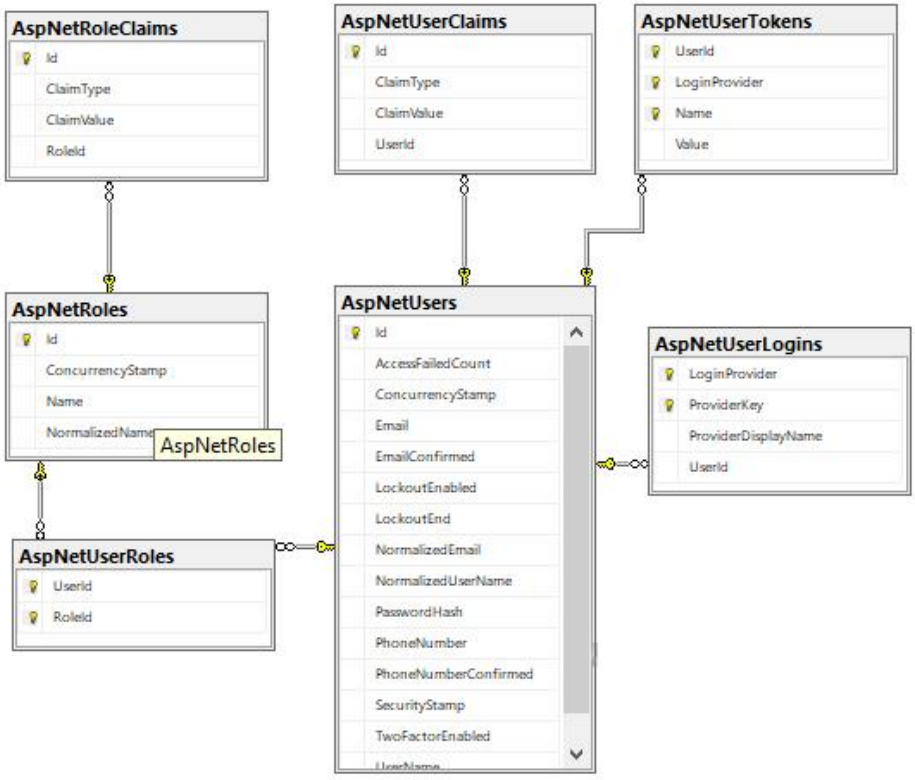

Рисунок 4.5 – схема авторизаційних даних користувачів

Наступна логічно обєднана сутність – шаблони.Дана сутність складається з наступних таблиць:

- TemplateTypes – типи шаблонів містить перелік назв елементів, для їх структуризації в рамках шаблону
- Template описує поточну одиницю шаблону її назву і за наявності код
- TemplateRelation дана таблиця служить для побудови графа залежностей між елементами, на основі того що кожний елемент знає про передній, а елементи верхнього рівня відповідно є шаблонами. Кожен запис в даній таблиці містить шлях по якому можна доступитись до нього в поточному звязку. І посилається на тип елементу самого проекту, щоб можна було структурно ідентифікувати поточний елемент.

Загальний вигляд представлено на рисунку 4.6.

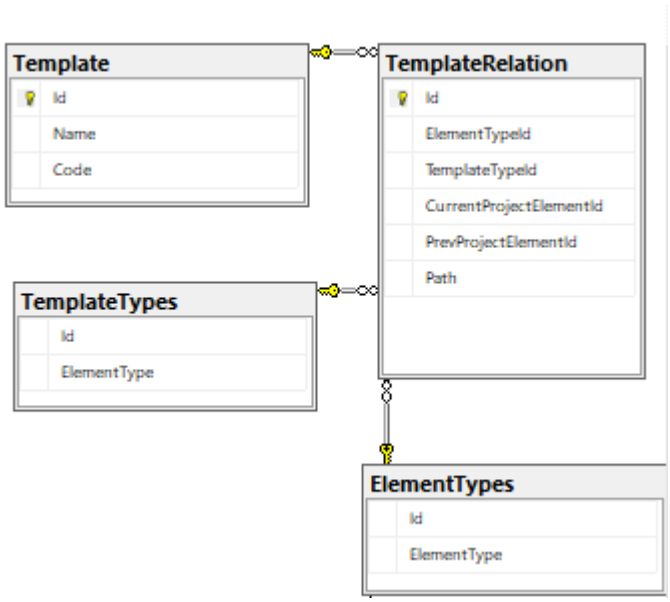

Рисунок 4.6 - схема даних шаблонів

Останім проте немало важливим елементом є логічна сутність, що відповідає за зв'язок між користувачем, його проектом і відповідно проектом і його складовими частинами.

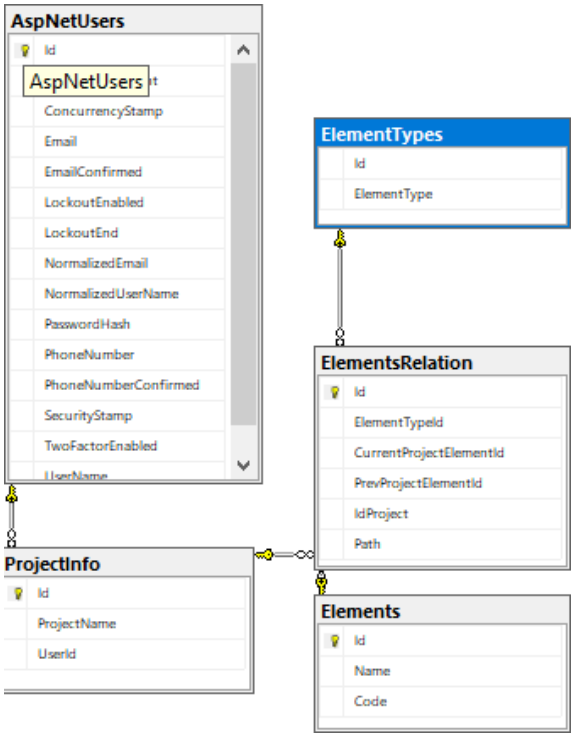

Рисунок 4.7 - схема даних користувацького проекту

Як видно з рис.4.7 та було сказано вище логічна сутність складається :

- TemplateTypes– типи елементів містять перелік назв елементів, для їх структуризації в рамках конкретного проекту
- Elements описує поточну одиницю проекту її назву і за наявності код
- ElementsRelation дана таблиця служить для побудови графа залежностей між елементами, на основі того що кожний елемент знає про передній, а елементи верхнього рівня відповідно є шаблонами. Кожен запис в даній таблиці містить шлях по якому можна доступитись до нього в поточному звязку. І посилається на тип елементу самого проекту, щоб можна було структурно ідентифікувати поточний елемент. Та верхні елементи посилаються на проект до якого належать.
- ProjectInfo описує інформацію про проект та якому користувачеві він належить.

4.2.2 Структура програмного рішення

Відповідно до описаного підходу в аналіз предметної області програмне рішення міститить три рівневу ахітектуру, що складається з користувацької частини побудованої як окреме рішення, що взаємодіє з кінцевим API серверу. Яке в свою чергу містить кінцеві точки що забезпечують доступ до реалізованої розприділеної логіки по шаблонізації рівня доступу до даних, бізнес-логіки і представлення для користувацьких додатків. Також дане рішення складається з свої поведінкових інтерфейсів в що являють собою бізнес-логіку даного додатку. А саме відповідають за створення копозиції заданих елементів для побудови графової структури з навігаційним словником пошуку.

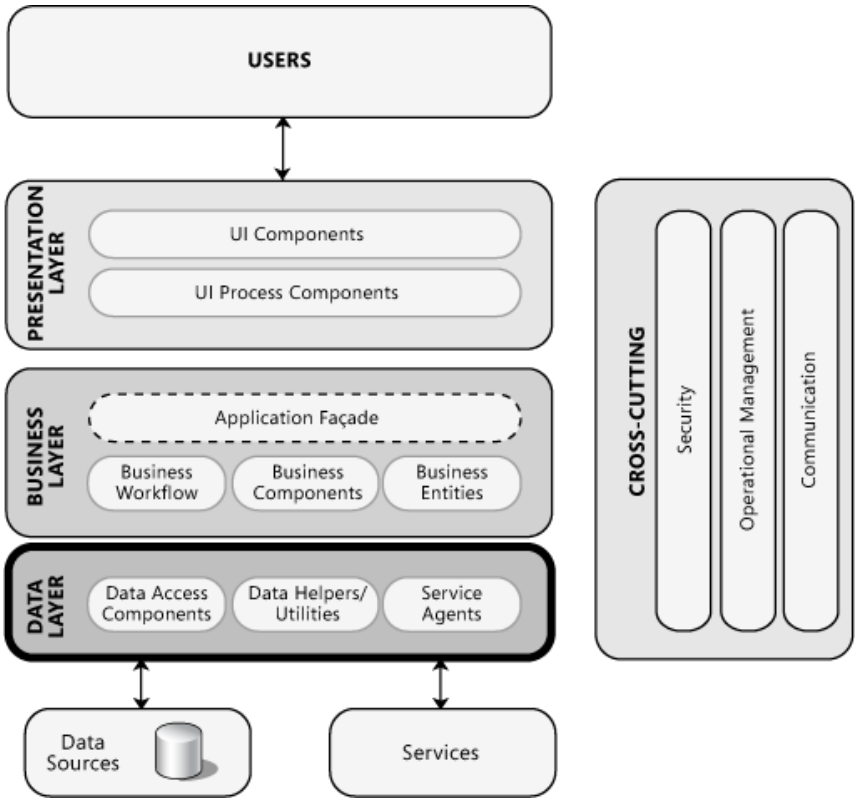

Рисунок 4.8 – трирівнева архітектура додатку

На рисунку 4.8 зображено загальний вигляд рішення в контексті трирівневої архітектури.

4.3. Опис програмного рішення.

Ключова ідея розробки полягає в тому щоб, ґрунтуючись на базовому понятті архітектури, побудувати реальну модель для шаблонізації процесу розробки додатків. Говорячи про поняття базової архітектури маємо на увазі модель, наведену на рисунку 4.9.

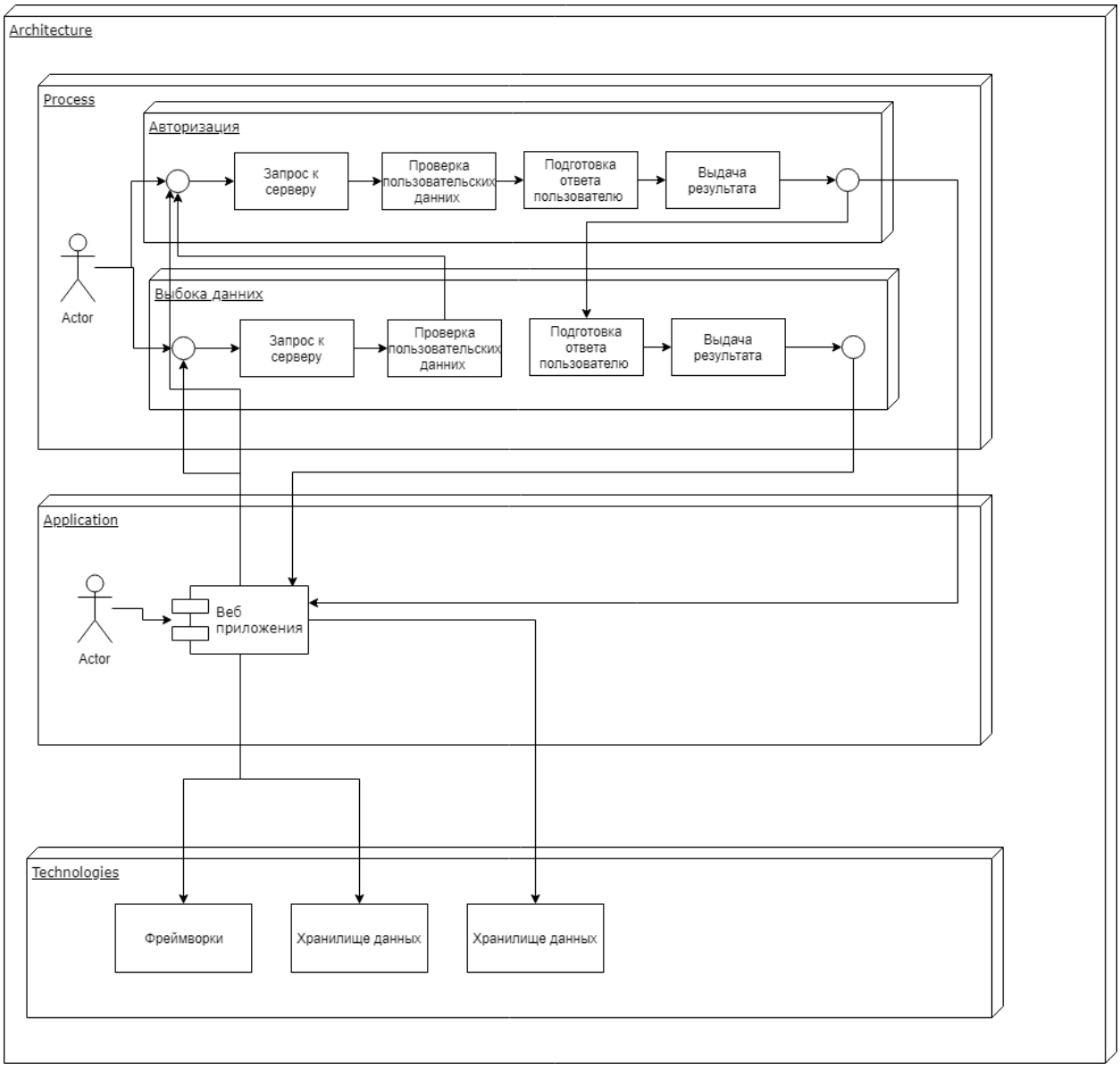

Рисунок 4.9 - базова архітектура системи

Як видно з наведеної моделі, є набір шаблонних процесів, які реалізуються одним застосуванням за рахунок засобів використання фреймворків і специфікацій тієї чи іншої технології. У свою чергу, застосування є багаторівневим і складається з шаблонних модулів. Схема оброблення запитів була наведена вище. Кожен такий модуль відповідно

будується з класів і інтерфейсів, що вже мають свої структурні одиниці і відповідний вигляд і методи в залежності від розв'язуваних задач згідно вимог користувачів.

Розглянемо процес шаблонізації більш детально. Проектування будь-якої системи розпочинається з реалізації бізнес-процесів і структури сховища даних, які будуть відповідати поставленим завданням. Згідно запропонованого підходу користувач системи зобов'язаний поставити модель схеми для бази даних, а саме описати сутності і вказати їх зв'язки.

Після цього етапу система будує базову функціональність у вигляді моделі даних скриптів на генерацію бази даних, рівень доступу до даних, а так само в ході створення моделі є зумовлені сценарії по роботі з користувачами і авторизацією, які користувач може вибрати в залежності від потреб.

Наступним етапом роботи користувача є генерація безпосередньо компоненту роботи з даними, а саме конструювання вибірки, підготовка даних з рядом умов за рахунок об'єднання умов з використанням запропонованого шаблону або створення власного.

Після даного етапу необхідно задати умови і умовний семантичний опис відповідних перетворень та обробки даних відповідно до поточного бізнес-процесу.

Наступним кроком є шаблонізація самого представлення з відповідним вибором та редагуванням найбільш підходящого шаблону з запропонованих.

Відповідно результатом є готовий додаток що згенерований внаслідок композиції генерації під модулів системи.

4.4 Приклади бізнес-процесів

Розглянемо основні бізнес-процеси, підтримка яких забезпечується будь-яким web-застосуванням.

Процес авторизації та оброблення даних схожі, точніше кажучи базуються на загальній концепції роботи веб додатку що зображена на рисунку 4.9. Це пояснюється тим, що різниця полягає, в першу чергу, в методах, що викликаються на рівні бізнес-логіки. Ще один аспект відмінностей пов'язаний

з тим, що ми розглядаємо конкретні приклади і можемо виділити ті чи інші інтерфейси класи і об'єкти відповідно до вимог користувачів. А загалом ще раз варто відзначити шаблонність будь-якого web-запиту. Розглянемо ще декілька прикладів бізнес-процесів. Наприклад при розробці системи оцінювання якості, швидкості, успішності, незмінним буде наявність розрахунку середнього арифметичного значення по тим чи іншим показникам. Для систем пов’язаних з обробкою заявок користувачів, будь то заявка на покупку, розсилку повідомлень, проходження курсів, буде агрегація даних заявок за обраним критерієм, такими як час подачі, умови подачі, користувач, тобто відправник.

Насамкінець, бізнес-процес по обробці ваучерів на знижки, що є також невід’ємною складовою цілого набору систем. Побудова даного бізнес-процесу ідентична іншим єдине що вирізняє реалізацію кожного бізнес-процесу це поведінкова стратегія на рівні бізнес-логіки, що реалізує дану задачу, в нашому випадку в залежності від умов що були задані під час створення даної компоненти ваучер(а точніше кажучи код для ваучера) буде генеруватись по різним схемам, але система гарантує унікальну видачу даного коду для кожного запиту.

4.4 Технологічні аспекти.

Ідея реалізації полягає в створенні веб інтерфейсу що дозволяє кінцевому користувачеві побудувати модель бази даних на основі якої генерується базовий каркас додатку а саме CRUD операції з даними с подальшою генерацією веб інтерфейсу в вигляді REST API. Наступним кроком є відображення даного загального вигляду додатку з можливістю генерації за допомогою семантичного задання зв’язків між відповідними моделями тієї чи іншої логіки поведінки додатку та реалізації бізнес-процесів необхідних даному додатку. Також варто відмітити наявність уже готових шаблонів які інтегруються в додаток з вказанням точок входу та виходу. Додатковою функціональністю є можливість створення та зберігання користувачем власних шаблонів а також можливість викладати їх в загальний доступ.

Формально інтеграція шаблону може відбуватись на різних рівнях архітектури, починаючи від метода закінчуючи інтеграцією цілої компоненти що складається уже з класів. Розглянемо детальніше дану модель. Кожен бізнес-процес складається з набору операцій, що необхідні для його реалізації. Кожна з таких функціональних одиниць містить відповідно вхід та вихід, що відповідає входу наступного блоку і тд. А також складається з вхідних та вихідних характеристик, для прикладу для сутності бази даних характеристичними властивостями є відповідні поля класу моделі. Спосіб побудови додатку, як зазначалось раніше є композицією модулів, що в свою чергу є композицією класів, що в свою чергу є композицією методів і тд. Відповідно формуючи вхідну сутність та її характеристику та знаючи необхідну вихідну сутність і її характеристику підібравши семантично необхідний метод або набір методів їх взаємо зв’язку отримуємо модель шаблонізації самого низького архітектурного рівня поданого в вигляді класів. На вищому рівні зберігається модель композиції, але поєднуються уже відповідні класи через виклики інших методів та генерації класів поведінки, що є вихідною точкою для модуля. Відповідно далі йде поєднання модулів в наслідок композиції їх вихідних точок та встановлення зав’язків рис.4.10.

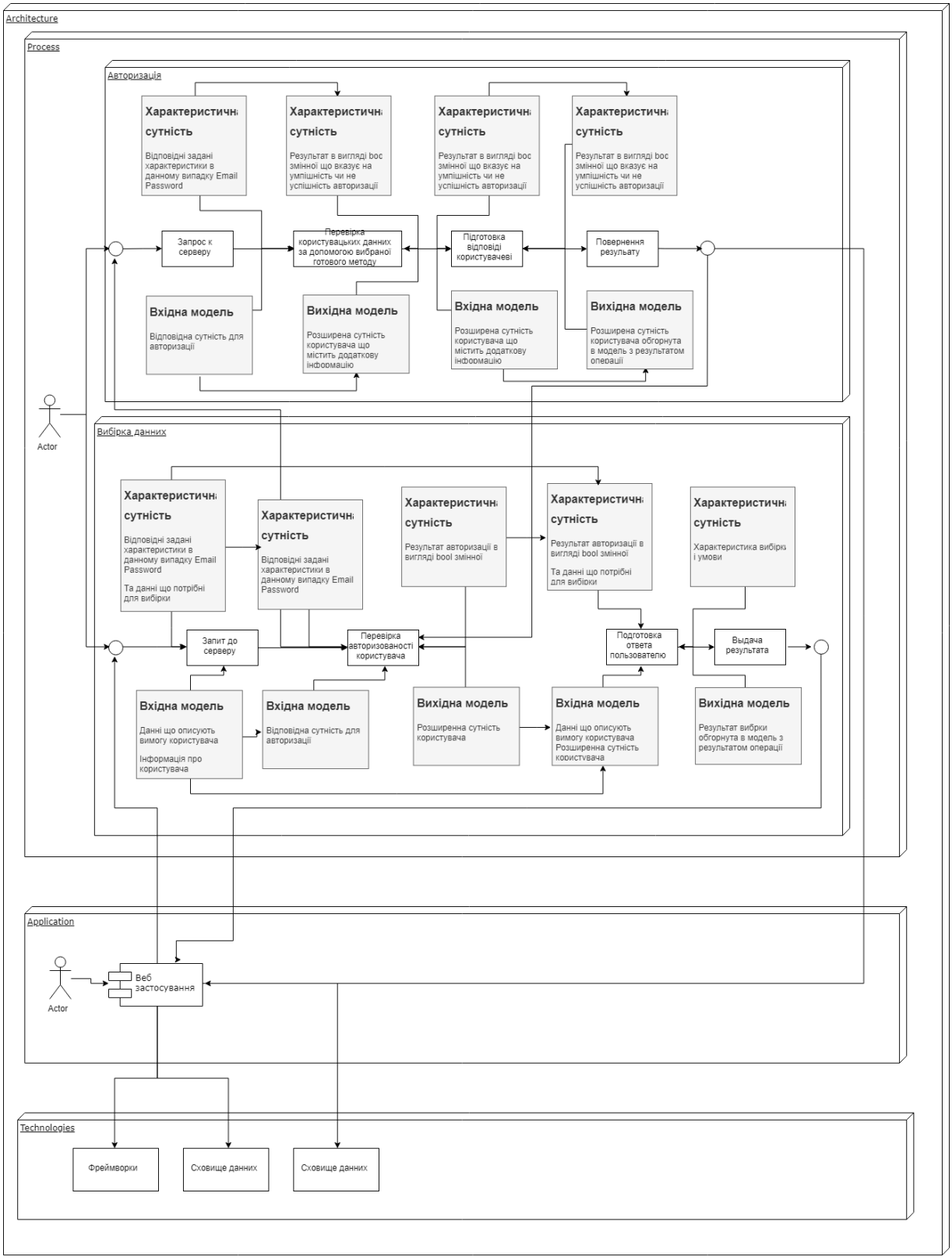

Рисунок 4.10 - композиція елементів бізнес-процесів

На рис. 4.10 представлено загальну модель семантичного запису, що дозволяє шаблонізувати два зв’язаних бізнес-процеси та відображена загальна архітектурна схема. Як видно із рисунку 4.10 задано два бізнес процеси, перший – авторизація користувачів, другій вибірка даних, що вимагає щоб

користувач був авторизованим. Розглянемо перший процес, відомо що з клієнтської частини буде звернення на сервер, з передачею моделі для авторизації, що містить логін і пароль. Наступним кроком є опис підпроцесу авторизації, для нього відома вхідна сутність та характеристична в вигляді мінімальної необхідної моделі, також відомий вигляд результату його характеристика - успішність чи не успішність операції авторизації, та в випадку успішності розширена модель інформації користувача. Відповідно наступний підпроцес на вхід приймає результат попереднього за рахунок чого між ними встановлюється зв’язок, тобто вихідна модель і характеристика попереднього процесу відповідає або є складовою вхідного набору наступного. Даний підпроцес описує формування результату, а саме створення обгортки, що описує результат операції як окрему сутність в якої у якості даних повертає модель користувача.

Другий бізнес-процес є складнішим. На вхід при запиті буде направлена інформація про авторизаційні данні користувача та необхідну інформацію для тієї чи іншої вибірки. Наступний підпроцес на вхід приймає частину даних і викликає шаблон вже реалізованого процесу авторизації та повертає результат, що є вхідними даними для підпроцесу вибірки, та моделі даного процесу розширяються даними необхідними для вибірки вказаними в вхідній моделі і відповідно до першого процесу будується та повертається результат.

4.5. Мовні аспекти.

4.5.1 Аргументація вибору серверу баз даних

Відповідно до розділу 1.5. Джерела даних за необхідності збереження даних вибір є між SQL та NoSQL базами даних. З урахуванням використання строго типізованої мови програмування для реалізації даної задачі та складних взаємо пов’язаних сутностей вибір SQL бази даних є очевидним. Відповідно до вище сказаного необхідно вибрати серед реляційних систем управління базами даних одну із запропонованих. В ході написання дипломної роботи було розглянуто:

- Microsoft SQL Server
- MySQL

SQL сервер більше, ніж реляційна СУБД

Головна перевага платного ПО в порівнянні з безкоштовним - це особлива підтримка, яку ви отримуєте. В даному випадку, перевага ще більш значуще, так як SQL сервер підтримується однією з найбільших компаній в світі. Microsoft створила додатковий інструменти для SQL сервера, які прив'язуються до реляційної СУБД, включаючи інструменти для аналізу даних. Система також має сервер звітів - Служба звітів SQL Сервера, так само як і інструмент ETL. Це робить SQL сервер швейцарським армійський ножем серед реляційних СУБД. Ви можете отримати подібні функції і в MySQL, але вам доведеться шукати в інтернеті сторонні рішення - що багатьом не підійде.

Система зберігання данних

Іншим великим розходженням між MySQL і SQL сервером, яке іноді упускають, це система зберігання даних. SQL сервер використовує єдину систему, розроблену Microsoft, в порівнянні з безліччю движків, пропонованих MySQL. Це дає розробникам, що використовують MySQL більше гнучкості, оскільки вони можуть вибирати різні системи для різних таблиць, грунтуючись на швидкості, надійності або якихось інших параметрах. Популярний движок MySQL - це InnoDB, який трохи втрачає в швидкості, але забезпечує посилену надійність. Інший відомий - MyISAM.

Скасування запиту

Кардинальною відмінністю між MySQL і SQL сервером є те, що MySQL не дозволяє вам скасувати запит в середині його виконання. Це означає, що, як тільки команда запущена на виконання, вам краще сподіватися, що будь-які збитки, який вона може зробити, є оборотним. SQL сервер, з іншого боку, дозволяє вам скасувати запит на півдорозі його виконання. Ця різниця може бути несуттєвим для адміністраторів, так як вони зазвичай виконують скрипти команд, і це рідко вимагає відміни під час їх виконання, чого не завжди скажеш про розробників.

Microsoft SQL Server - система управління реляційними базами даних (СУРБД), розроблена корпорацією Microsoft. Основна використовувана мова запитів - Transact-SQL, створена спільно Microsoft і Sybase, вона використовується для роботи з базами даних розміром від персональних, до великих баз даних масштабу підприємства.

MySQL безкоштовний і підходить більше для завданнь, де структура БД не велика, для невеликих сайтів, веб-додатків. MS SQL більш потужна, розширення sql-а (T-SQL) дозволяє вирішувати набагато більше завдань. Також має безкоштовну версію. Через переваги в швидкодії та стабільності роботи я обрав MS SQL Server як СУРБД.

4.5.2 Аргументація вибору технології розробки

Для створення серверної частини веб-додатку, на поточний момент часу міститься безліч технологій, до основних можна віднести C++, PHP, Java, .Net, Python, Node JS, відповідно до діаграми зображеної на рис. 4.11.

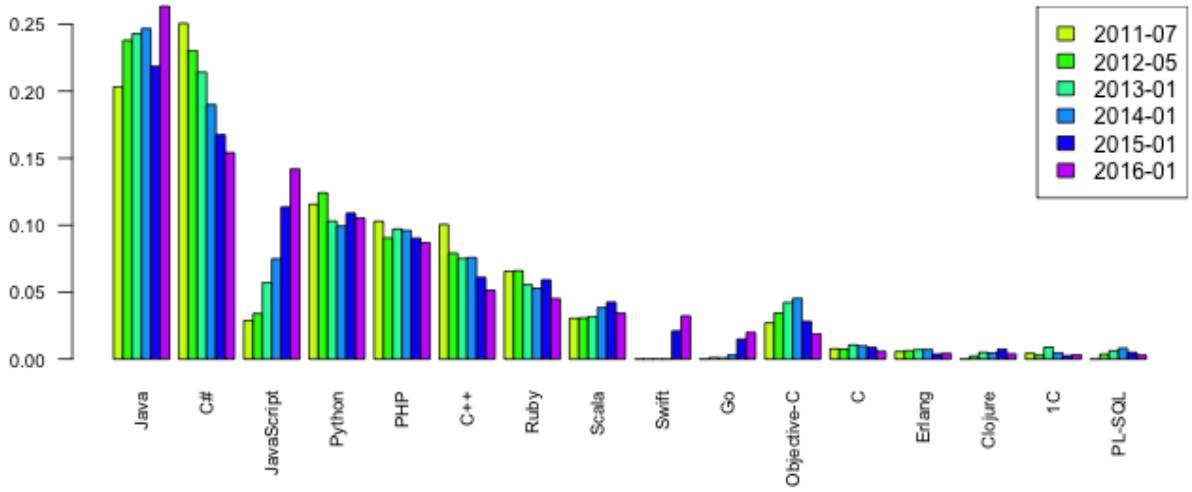

Рисунок 4.11 – ключові технології

Розглянемо кожну відповідно технологію з запропонованих альтернатив детальніше, визначимо її переваги та недоліки провівши огляд концепцій розробки. Сучасні мови програмування поділяються на низько рівневі та високо рівневі мови.

C++

В класифікації розміщена між високорівневими та низькорівневими мовами. Дана мова є обєктноорієнтованою та містить обєктно орієнтований та процедурний підходи. Є гнучкою платформою для створення застосунків різного типу, починаючи від простих консольних закінчуючи складними веб-орієнтованими системами. Характиристичною особливістю, що не відносить її до високорівневих мов є пряма робота з пам'яттю, завдяки чому дана технологія має як переваги так і недоліки. До переваг можна віднести швидкодію. До недоліків часові затрати на створення додатків, оскільки пряма робота з памяттю потребує ряду додаткових обробників, що вже інтегровані в мови що базуються на байт коді.

Відповідно, можливості по реалізації значно більші в межах низькорівневих задач, до яких можна віднести системне програмування, програмування контролерів та мікропроцесорної техніки. Відповідно до поставленої задачі, її реалізація на c++ витратить набагато більше часу на фоні реалізації на високорівневій мові на кшталт JAVA .Net. Порівняння швидкості опрацювання одного і того самого запиту сервером реалізованим на c++ та ASP.Net зображено на рис. 4.12.

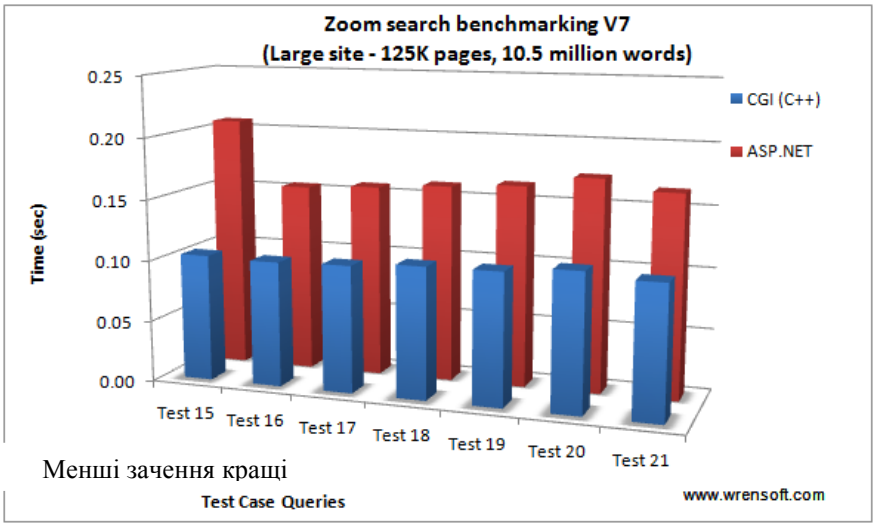

Рисунок 4.12 – порівняння швидкодії реалізацій серверних рішеньCGI та ASP.NET

PHP

Високорівнева мова, що ключає в себе всі необхідні інструменти для розробки. Ключовим недоліком є відсутність підтримки MVC в рамках платформи, що негативно відображається на зручності розробки, гнучкості готової системи. Дана технологія в якості розширення своїх функціоальних можливостей пропонує ряд реалізованих над платформою фреймворків по розмежуванню логіки від представлень. Дана мова призначена для вирішення стандартних задач, рядом стандартних способів в випадку відсутності необхідності створення високо-продуктивної системи.

Порівняльна характеристика швидкості відгуку за запитом зі зверненням в базу даних наведена на рис. 4.13.

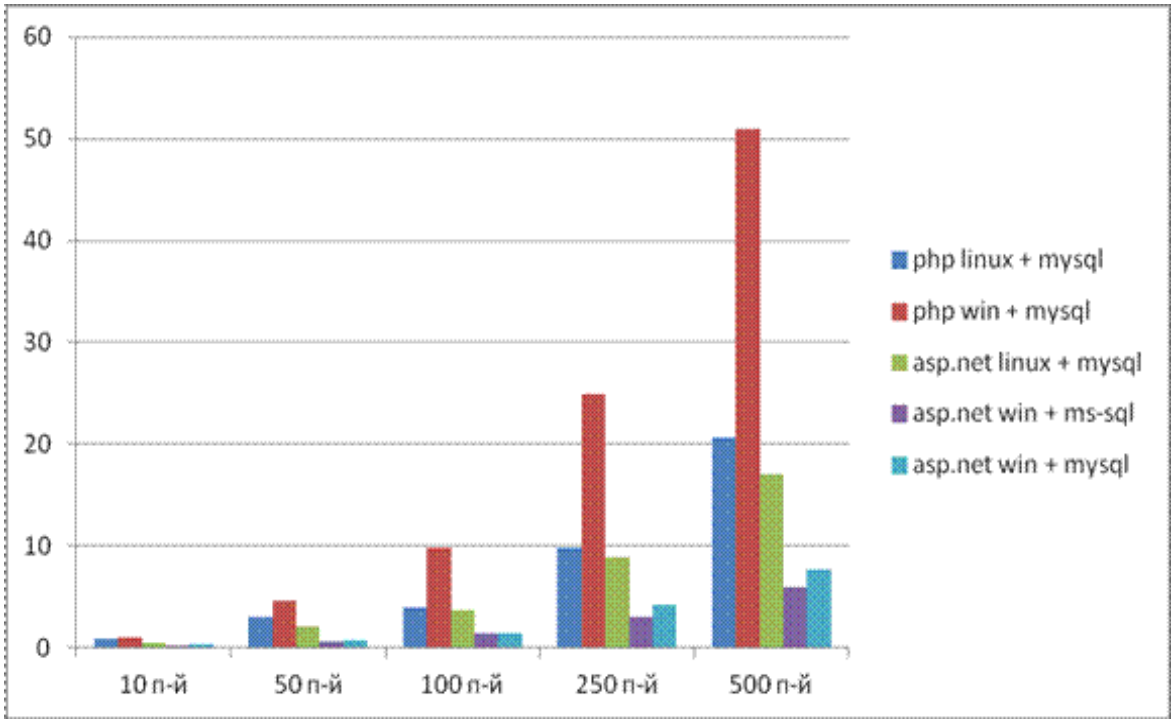

Рисунок 4.13 – Порівняння швидкості відгуку веб-додатків PHP ASP.NET

Java

Java є основним конкурентом .Net в межах створення веб-застосувань.

JSP

Технологія Java Server Pages (JSP) є компонентом єдиної технології створення застосувань що базуються на J2EE, з використанням web-інтерфейсу. JSP – альтернатива технології Java Servlet, методологія створення

програм, здатних динамічно генерувати відповідь на поточний запит до застосунку. До моменту відображення відповіді користувачеві після отримання запиту сервером і обробкою поточного запиту застосуванням, вбудована процедура обробить результат виконання та конвертує його у сервлет. Сервлет є конкретною реалізацією заданого інтерфейсу, що описаний на мові Java. Сервлет існує в межах того чи ншого веб контейнеру, що розміщений в веб-застосуванні. Web-контейнер реалізує комунікацію клієнтів та функціональних одиниць веб-застосування. Кожен сервлет містить свій опис в окремому класі, який реалізує відповідну абстракцію Servlet.

Дана технологія аналогічна до ASP Web Forms, на разі її застосування обмежене підтримкою вже існуючих проектів. На зміну даній технології прийшло рішення в вигляді фреймворку Spring, що містить розмежування логіки та представлення. Причому варто відмітити належність даного фреймворку до архітектурно коректних, оскільки він побудований з урахуванням принципів S.O.L.I.D., що надає гнучкості коду. Даний факт в межах архітектури додатку надає можливість маштабування. Порівняльна характеристика фреймворків наведена на малюнку 4.14.

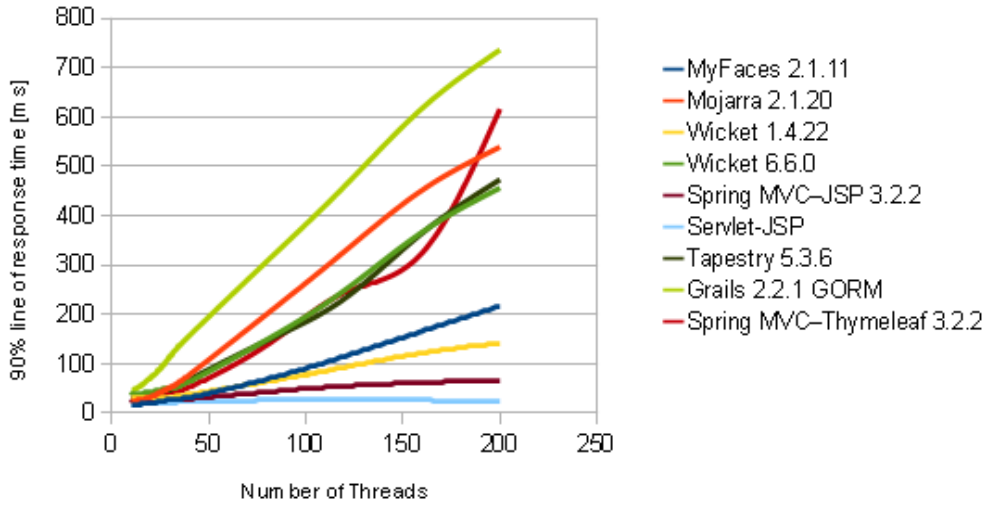

Рисунок 4.14 – порівняння фреймворків

Spring

Spring Framework може бути використаним будь-яким додатком Java, але основним його призначенням є розширення платформи в межах створення

веб-застосувань обсумовлених платформою Java EE. Незважаючи на це, Spring Framework не вимагає конкретної моделі програмування. Даний фреймворк став популярним в спільноті Java в межах доповнення моделі Enterprise JavaBean (EJB).

Spring Framework містить набір модулів, що забезпечують можливість реалізації більшості задач зручним способом.

Аспектно-орієнтоване програмування: забезпечує можливість програмування наскрізних процедур

Доступ до даних: робота з джералами даних. В межах баз даних є можливість взаємодії з реляційними і NoSQL базами даних, відповідно з використанням технологій JAVA по взаємодії з сховищами таких як : JDBC.

Транзакційність: композиція набору сервісів відповідно до сервісно орієнтованох архітектури.

Реалізація шаблону MVC: можливість розмежування логіки від представлення, що забезпечує гнучкість коду.

Авторизація і аутентифікація: набір готових рішень по взаємодії з описаними протоколами аутентифікації користувачів, стандартними методами джава, що доповнені в даному фреймворці.

Серед мінусів даної технології відносять спосіб організації контейнерів. В ході огляду даного фреймворку можна виділити його найблищого конкурента ASP.MVC в .Net. В ході порівняння даних технологій можна зазначити ряд параметрів по яким Spring суттєво відстає від конкуренту.До них відносять: ОЗУ та швидкість відклику. Порівняльна характеристика наведена на рисунку 4.15.

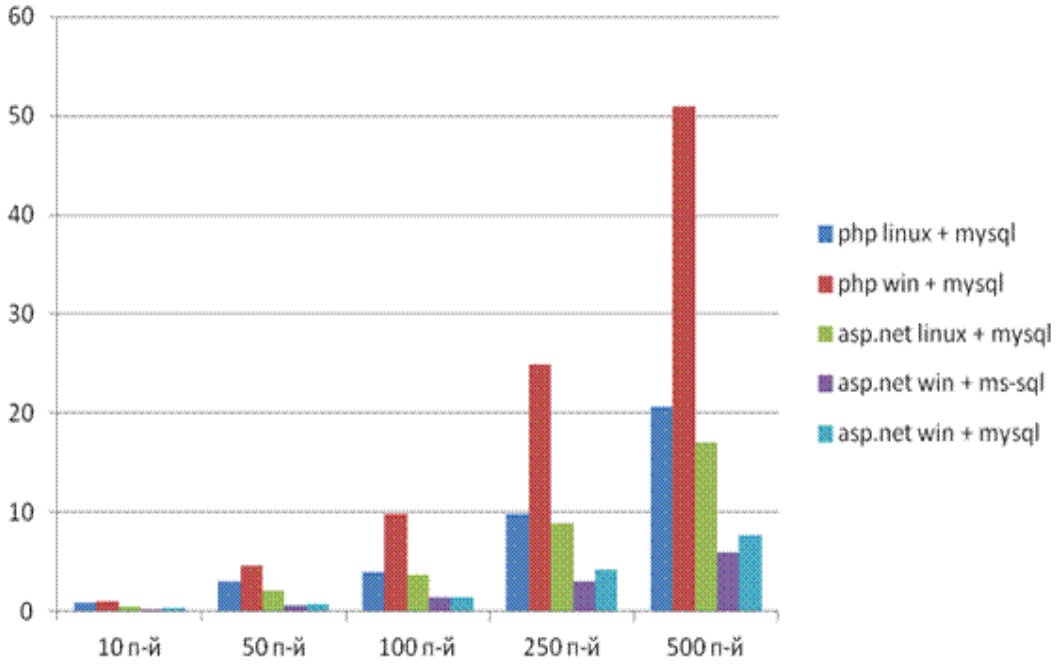

Рисунок 4.15 – порівняльна зарактеристика Spring ASP MVC

Node JS

Node.js платформа, що має велику динаміку розвитку. Через цілий ряд чинників:

- відкритий код;
- асинхронне опрацювання запитів;
- відсутність блокування потоків вводу/виводу;
- модульність;
- продуманий пакетний менеджер;

Мова, що застосовується в даній платформі як слідує з назви JavaScript.

Дана технологія дозволяє створювати продукти різної складності, проте є орієнтованою під написання високопродуктивних веб-застосувань. З використанням вказаних переваг та побудови технології на онснові не блокуючих викліків, за допомогою графової структури обробників дана технологія є конкурентно спроможною, проте мітить ряд недоліків. Ключовими недоліками є високий поріг входження оскільки обробка запиту проходить низько-рівнево. В межах поточної задачі ключовим недоліком є

відсутність типізації, що негативно сказується при роботі з строготипізованими джерелами даних.

.Net

Microsoft .NET — програмна технологія, що була запроваджена фірмою Microsoft, в якості платформи для створення, віконих, консольних та веб-застосувань невеликої складності, так і для корпоративних веб систем. Існує багато чиників завдяки яким можна сказати що дана платформа є продовженням концепції і принципів закладнеих в технології Java. Композиція служб, застосунків написаних будь-якими мовами в рамках розглянутої в розділі 2 концепції сервісно орієнтованої архітектури є однію з ключових ідей платформи .NET. Варто відмітити, що в незалежності від реклами, що вибудувує хибну думку про те, що даний підхід існує виключно в даній платформі, платформа Java має таку саму можливість.

Кожна модульна частина, що називається бібліотекою (збіркою) в платформі .NET містить інформацію про свою версію, що в свою чергу запобігає появі конфліктів версій.

.NET — крос-платформова технологія, на даний момент часу існує реалізація для наступних операційних систем Microsoft Windows, FreeBSD (від Microsoft) і ОС Linux. Авторські права даної технології відносятся до створення середовищ виконання (CLR — Common Language Runtime) для програмних засобів платформи .NET. Завдяки такій політиці авторського права, IDE для .NET випускаються низкою фірм для різних мов вільно.

.NET умовно можна поділити на дві основні частини — середовище виконання (по суті віртуальна машина) та інструментарій розробки.

До середовищ розробки .NET-програм відносять SharpDevelop, Visual Studio .NET (C++, C#, J#), Borland Developer Studio (Delphi, C#) тощо. Також данну технологію розробки програмних засобів було інтегровано в Eclipse. Присутня можливість створення програмних засобів даною технологією за рахунок використання текствого редактору та компідяції через виклик консольних команд.

Схожістю з Java в рамках середовища розробки є створення обома технологіями байт-коду, що призначений та виконується на відповідній віртуальній машині, що забезпечує кросплатформеність в результаті компіляції байт коду в машиний код з використанням тієї чи іншої реалізації віртуальної машини під ту чи іншу операційну систему відповідно. В .NET вхідною мовою даної машини є CIL (Common Intermediate Language), яку зачасту називають в джерелах, як MSIL (Microsoft Intermediate Language), або просто IL.

Варто відмітити, що одина з перших реалізацій JIT-компіляторів для Java була теж розроблена фірмою Microsoft (на поточний момент часу в Java використовується покращена багаторівнева компіляція — Sun HotSpot). Сучасний підхід до створення програмних засобів за допомогою високо рівневих мов програмування та технологій дозволяє досягнути наближеного рівня швидкодії з традиційними «статичними» компіляторами (наприклад, C++).

Окремо в якості переваг слід розглядати наступні можливості тенології розробки програмних засобів на платформі .net для розробника:

- наявність делегатів та подій,
- можливість асинхроного виконання задач через async Thread Pool всередині ядра, що підвищує швидкодію додатку при багатопоточній роботі та великій інтенсивності, а також збільшує відмовостійкість,
- наявність Lambda виразів, що є замиканнями конкретно визначену в тілі виразу змінну, певного типу, що є елементом колекції та забезпечує спрощення механізму опрацювання структур даних
- наявність LINQ запитів, в межах LINQ To Entities, LINQ To Objects, LINQ To SQL.

Про веб-технологію розробки Asp Web Forms відповідно до наведеної інформації в розділі 1 можна зробити наступний висновок.

Дана технологія є застарілою та була реалізованна Microsoft, в якості веб-аналогу віконої технології розробки win form, що вимагало постійних звернень на сервер з повною відправкою сторінки її стану. Також мінусом є те що клієнтська-логіка виконувалась на сервері, що значно сказалося на ефективності додатку в порівнянні з ASP MVC.

ASP MVC виходячи з опису в першому розділі - технологія фундаментом якої є шаблон MVC, що надає розмежування логіки від представлення.

Завдяки цьому реалізується концепція розподілу логічних функцій та незалежність компонентів, у зв'язку з чим спрощується реалізація розробки окремих компонентів архітектури. Завдяки даному розмежуванню інтеграція в процес з цілю тестування окремих модулівв значно спрощена через можливість підстановки MOQ об’єктів.

Імплементація самого шаблону може відрізнятись, але чітко обумовлює розмежування відповідальності та забезпечує відсутність жорстких звязків. В елемента шаблонізації клієнтської сторінки - Razor, що значно випереджає ASPX шаблонізатор через кращу структурованість коду та іделогічно інший підхід до обробки сторінок. Також дана технологія містить можливість створення REST API, що надає можливість використовувати сервер тільки для опрацювання запитів і перекласти відповідальність за побудову складних представлень на сучасні фронтенд технології чим зменшити навантаження на поточний сервер.

.Net Core MVC технологія у котру ввійшли переваги звичайного ASP MVC разом з допрацьованою архітектурою та кросплатформеністю. На основі якої побудований сервер як REST API.

4.5.3 Аргументація вибору технології для реалізації клієнського додатку

Клієнтське представлення є зовнішнім виглядом системи, що у випадку веб-застосувань є веб сторінкою, що відображається в браузері. Браузер сприймає мову розмітки HTML та стилізує відповідне відображення за допомогою каскадних таблиць стилів. За програмну роботу з сервером відповідає JavaScript. Оскільки тенденція поточного розвитку технології

полягає в винесенні максимальної кількості логіки на сторону клієнта, то спочатку зявилися окремі зовняшні модулі, що інкапсулювали в собі ту чи іншу шаблону поведінку в залежності від сфери застосування. Наступним кроком була композиція даних модулів в готові фреймворки. Перші кроки :

- Angular.js
- Backbone
- Knockout

Проте в всіх вище зазначених технологіях були архітектурні прорахунки, що зумовлювались спробами композиції фреймворків як структурних елементів, а не графових компонентів. В ході розвитку даних технологій було створено ряд оновлених фреймворків таких як Angular, React. Кожен з них містить допрацьовану архітектуру та концептуально інший підхід до створення клієнтської веб-частини. В їх склад ввійшли шаблонізатори для динамічної побудови DOM елементів, вибудовано концепцію залежності кожної бібліотеки від інших, вирішено проблеми версійності.

Розглядаючи кожен із фреймворків можна виділити його сферу застосування. React зручний в випадку підтримки багатьох станів сторінки і відповідної реакції на їх зміну. Angular надає більш комплексне рішення, але направлений на створення клієнтського застосування загалом без конкретної привязки до сфери застосувань та типу додатку. Даний фреймоврк містить в собі всі функціональні можливості Reacty та доповнює їх.

4.5.4 Перелік використаних технологій

Відповідно до аргументації наведеної в попередніх розділах технології, що будуть використані наступні:

- Клієнтська частина Angular 7.
- Серверна частина .Net Core MVC (REST API).
- СУРБД MSSQL Server.

4.6. Висновки

В даному розділі аргументовано вибрану архітектуру технології. Розглянуто та аргументовано вибір серверу застосунків, веб-серверу. Описано алгоритм роботи програмного засобу. Аргументовано вибір серверу баз даних, наведено загальну структуру бази даних для зберігання інформації необхідної для генерації веб-застосувань. Аргументовано мовні аспекти в межах вибору технологій розробки серверної та клієнтської частини.

5. ЕКСПЕРЕМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

5.1 Завдання експерементального дослідження

Задача, що відображена в даному розділі полягає в шаблонізації створення веб-додатку. Даний додаток має відповідати наступним вимогам:

- Містити авторизацію та регістрацію користувачів
- Виконувати розсилку електроних листів в рамках про успішність реєстрації
- Генерувати промокоди
- Містити зручний та простий інтерфейс для управління

5.2 Створення архітектури бази даних додатку

Першим кроком в запропонованому рішенні для автоматизаці процесу створення веб-застосувань є створення архітектури бази даних. Відповідно використовуючи веб-інтерфейс системи системи створивши в ньому новий проект, відбувається перехід до сторінки створення архітектури бази данних. Після чого користувач має можливість задати схему додаючи і заповняючи кожну таблицю, вказавши її назву та додаючи її колонки вказуючи їм тип. Користувачеві доступні можливості редагування та видалення відповідних елементів таблиці і самої таблиці. Видалення елементу відображено на рис.5.2, видалення таблиці на рис.5.3, редагування елементу рис.5.4. В правій частині екрану є можливість застосування шаблону. Для поточної задачі міститься один шаблон, що відповідає за авторизацію, при його виборі, автоматично буде додано необхідні таблиці в БД. Також є можливість завантаження опису таблиць в вигляді класів та JSON файлу. Зовняшій вигляд екрану відображено на рис.5.1.

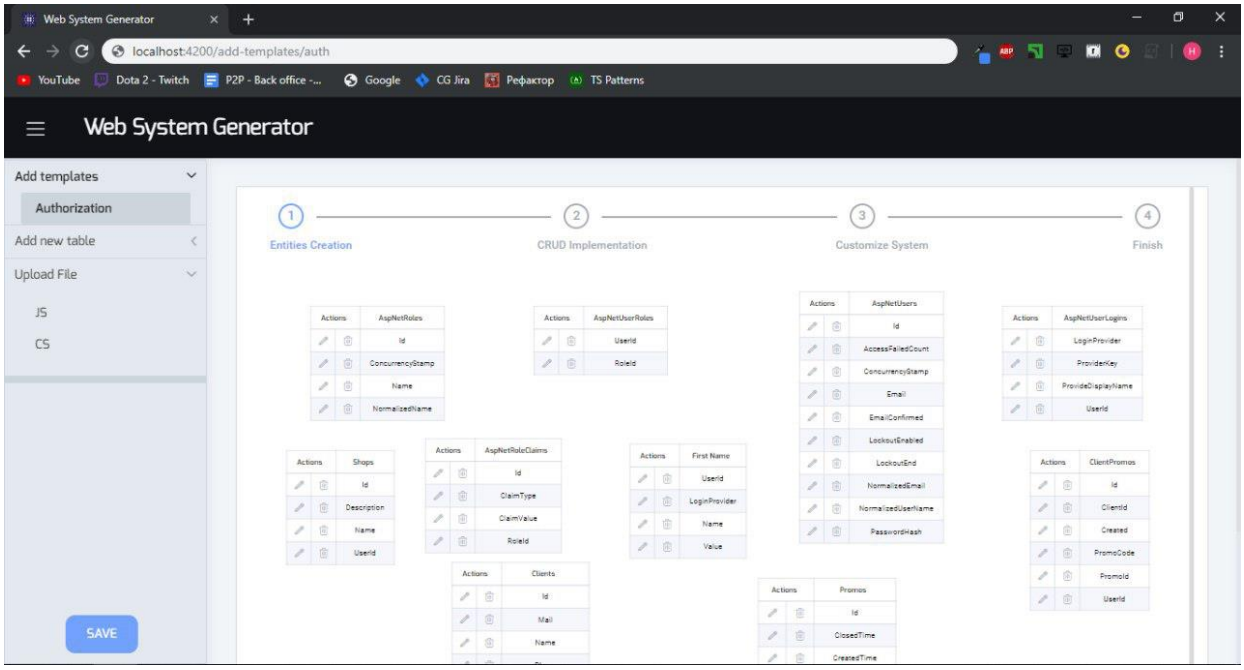

Рисунок 5.1 – створення структури даних

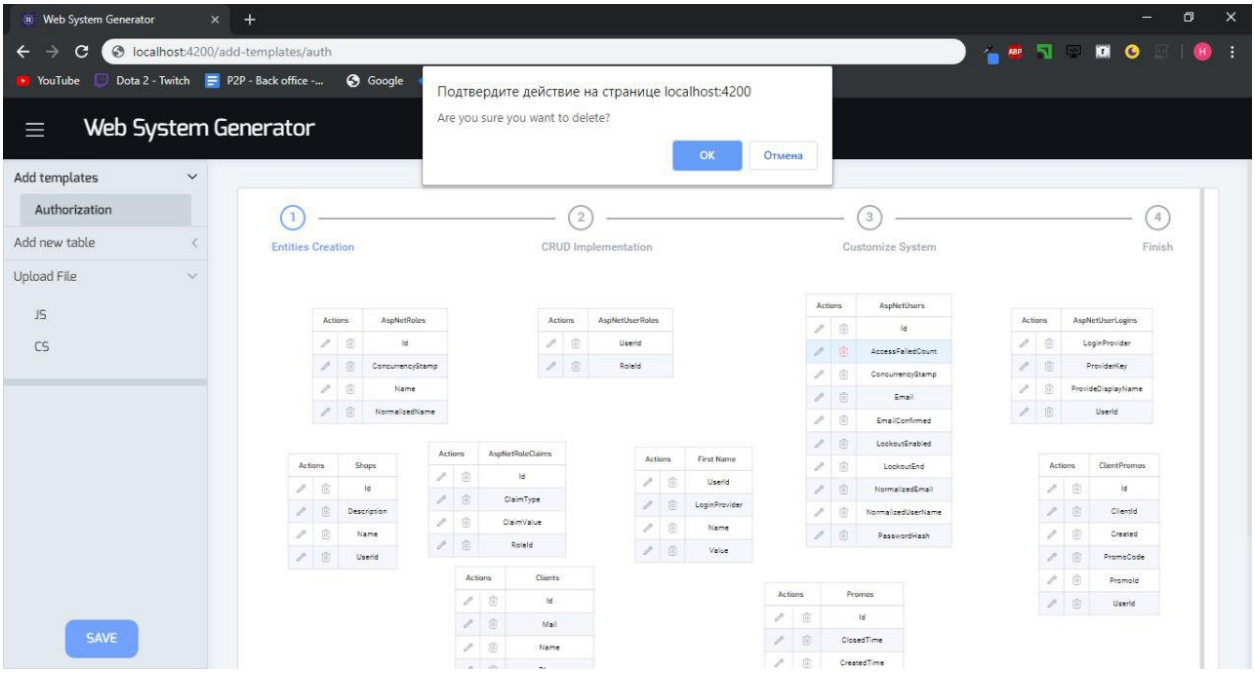

Рисунок 5.2 – видалення таблиці

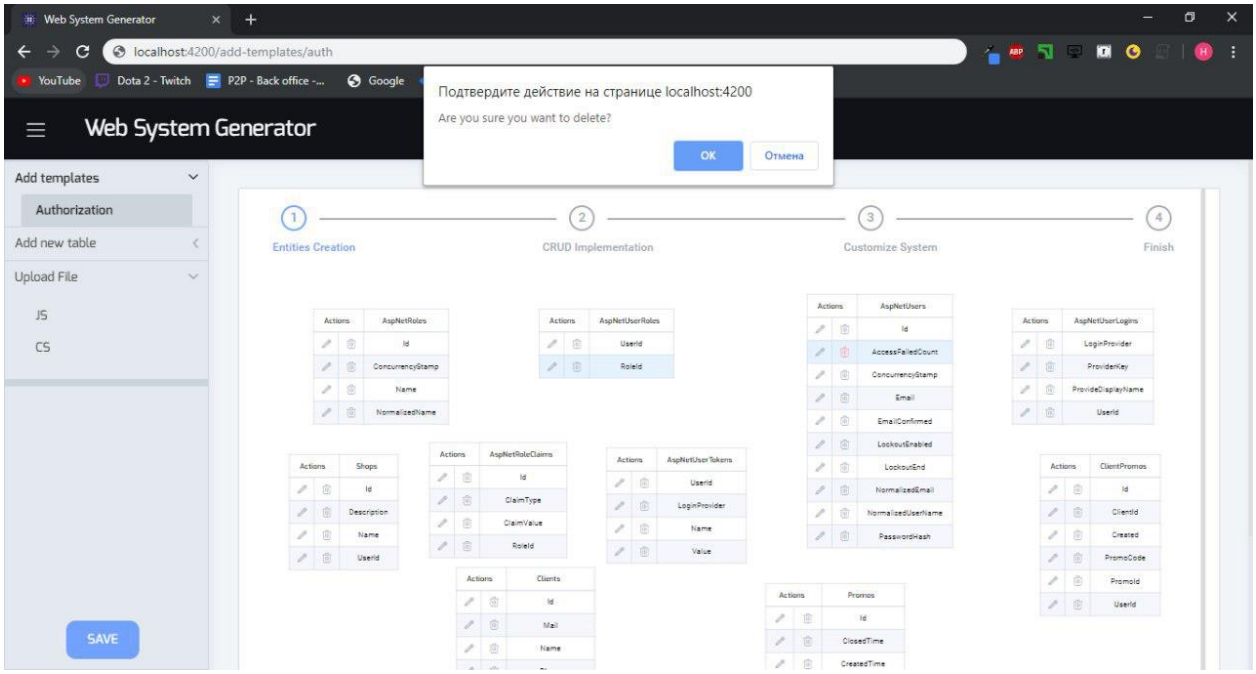

Рисунок 5.3 – видалення поля таблиці

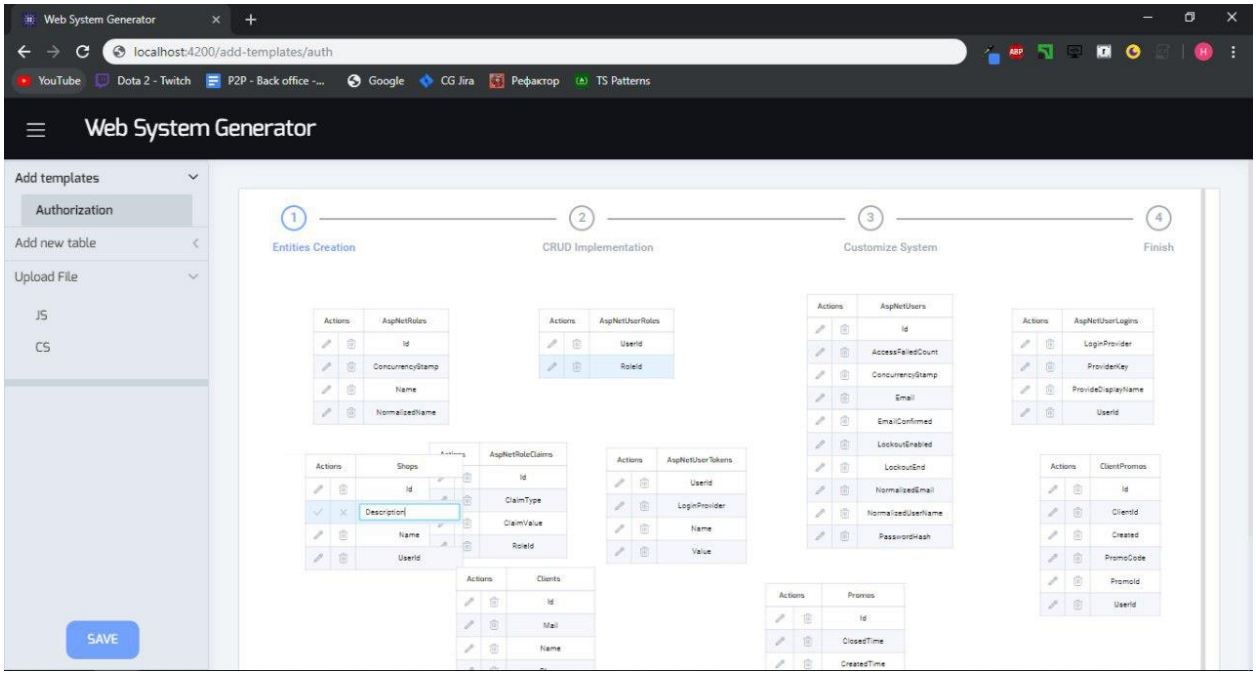

Рисунок 5.4 – редагування поля таблиці

5.3 Генерація коду

Відповідно до описаного в розділі 3 алгоритму та вказаного в розілі 4 технічного рішення, система відслідкувавши та зберігши кожен крок користувача переходить до єтапу побудови веб-застосування, що містить CRUD операцій з вказаними сутностями бази даних, всіх рівнів системи згідно з алгоритмом опрацювання даних, що наведений в розділі 4.

5.4 Доповння функціональними можливостями

Пілся кроку генерації CRUD операцій отримано систему з усіма рівнями архітектури, що відображається користувачеві, як продемонстровано на рис.5.5. Відображено найвищі структурні елементи архітектури з наступним рівнем вкладеності елементів в них. Відповідно користувач має можливість вносити до них зміни. При виборі певного компоненту система з права відображає перелік шаблонів що доступні для інтеграції в даний архітектурний компонент. При виборі складової частини компоненту, система сприймає його як верхній рівень та відображає його складові частини відповідно до підходу декомпозиції елементів. В випадку вибору стратегії генерації випадкових чисел користувачеві буде відображено зв’язаний з нею інтерфейс та сигнатуру методу генерації самого числа, що відображено на рис.5.6.

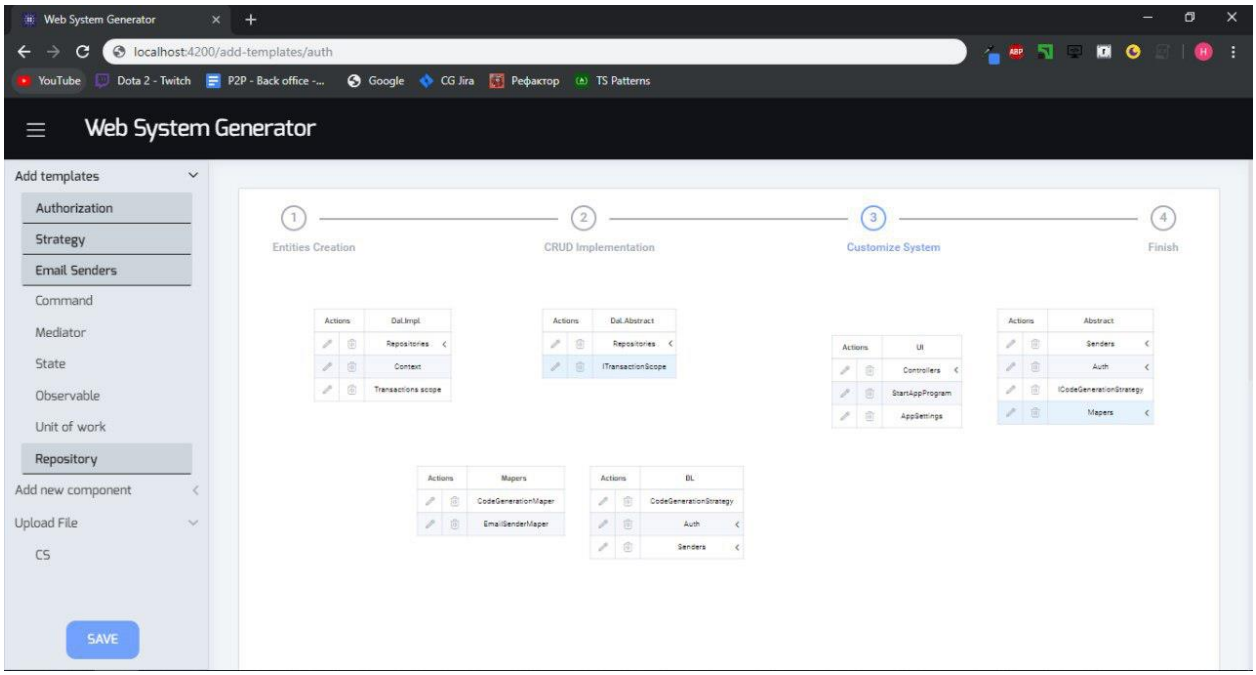

Рисунок 5.5 – загальний вигляд компонентів верхнього рівня

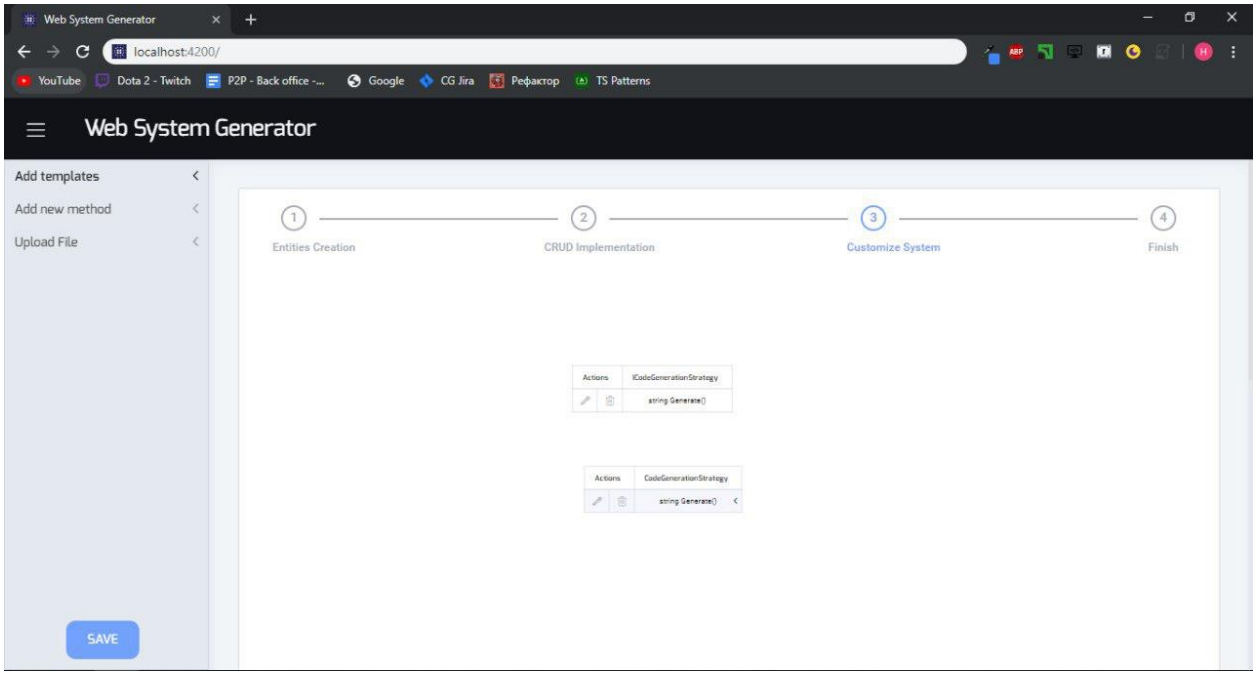

Рисунок 5.6 – загальний вигляд класу як обраного компоненту

5.5 Висновки

В даному розділі було розглянено роботу програмного продукту, що забезпечує швидке створенняя веб-застосувань. Наведено ключові користувацькі екрани по роботі зі створення застосунку. Описано принцип роботи та наведено послідовність дій в ході використання даної технології. Результатом є створений програмний продукт, що включає в себе всі необхідні шари архітектури.

6 РОЗРОБОКА СТАРТАП-ПРОЕКТА

6.1 Опис ідеї проекту

Описані в магістерській роботі підходи до генерації коду систем з веб представленням можна застосувати для розробки широкого кола інформаційних систем. За ідею для стартап-проекту була вибрана технологія генератор веб систем з можливістю подальшої їх модифікації.

Таблиця 6.1 - Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Технологія генератор веб систем	1. Ітеграція в процес розробки проектів з веб представленням приватними особами, малими, середніми та великими компаніями	1. Зкорочення часу, що використовується на розробку кожного базового сервісу в рамках складного рішення 2. Можливість детального налаштування кожного компоненту системи

Таблиця 6.2 – Визначення характеристик ідеї проекту

п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	SimplCommerce	WordPress			
.	Вартість експлуатації	5800 грн/місяць	12 000 грн/місяць	6000 грн/місяць	-	-	+
.	Вартість обслуговування	6500 грн/місяць	12 000 грн/місяць	6 000 грн/місяць	-	-	+
.	Вартість знижки	2 грн/послугу	0,01 грн/послуг	47,4 грн/послугу	-	+	-
.	Кросбраузерність/Кросплатформеність	Google Chrome 35.0+, Mozilla Firefox 35.0+, Safari7+, IE10+, Opera11+	Google Chrome 35.0+, Mozilla Firefox 40.0+, Safari7+, IE10+, Opera11+	Windows Vista, 7, 8 10 Mac OS X Lion (10.7)	-	+	-

Продовження таблиці 6.2

.	Інтеграція із смартфоном	iOS7.1.2+, Android4.0+	iOS7.1.2+, Android4.0+	Відсутня	-	-	+
	Робота в мережі	Присутня	Присутня	Відсутня	-	+	-
	Робота в локальній комп'ютерній мережі	Відсутнє	Відсутнє	Присутнє	-	-	+
	Формат універсальний вказівник ресурса (URL)	Стандарт W3C	Стандарт W3C	Відсутній	-	+	-
	Час завантаження сайту	0.5 с	0.3 с	0.5 с	-	+	-
0	Система керування контентом	Стандарт RSS	Стандарт RSS	Стандарт JSR-170	-	+	-
1	Завершеність (вірогідність відмови)	Висока	Середня	Висока	-	+	-

Продовження таблиці 6.2

2	Стійкість до відмов	Середня	Висока	Висока	-	+	-
3	Наявність системи резервного копіювання	Присутня	Присутня	Присутня	-	+	-
4	Відновлюваність	Висока	Висока	Висока	-	+	-
5	Технологічна собівартість	52 000 грн	112,5 грн	56 000 грн	-	+	-
6	Легкість освоєння	Середня	Середня	Середня	-	+	-
7	Наявність методичних вказівок для використання	Присутні	Присутні	Присутні	-	+	-
8	Автоматична обробка заявки на сервіс	Присутня	Присутня	Присутня	-	+	-
9	Наявність служби технічної підтримки	Присутня	Присутня	Присутня	-	+	-

Продовження таблиці 6.2

0	Автоматична система оплати сервісу	Присутня	Присутня	Присутня	-	+	-
2	Технічна документація	Присутня	Відсутня	Відсутня	+	-	-
3	Надання інтерфейсу для зовнішнього програмного забезпечення	Присутнє	Відсутнє	Відсутнє	+	-	-
4	Інтеграція з платформами інших сервісів	Присутнє	Частково присутнє	Відсутнє	+	-	-
5	Генерація унікального ID номера користувача	Присутнє	Відсутнє	Відсутнє	+	-	-

6.2 Технологічний аудит ідеї проекту

Таблиця 6.1 - Технологічна здійсненність ідеї проекту

№ п/п	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1		REST (Representational State Transfer)	Наявні	Доступні
2		SOAP (Simple Object Access Protocol)	Наявні	Доступні
3		XML-RPC (Extensible Markup Language Remote Procedure Call)	Наявні	Доступні
Обрана технологія реалізації ідеї проекту: REST (Representational State Transfer)				
4		Basic access authentication	Наявні	Доступні
5		OAuth	Наявні	Доступні
6		За допомогою токенів	Наявні	Доступні
7		На основі сертифікатів	Наявні	Доступні
Обрана технологія реалізації ідеї проекту: За допомогою токенів				

6.3 Аналіз ринкових можливостей запуску стартап-проекту

Таблиця 6.2 - Попередня характеристика потенційного ринку проекту

п/п	Показники стану ринку (найменування)	Характеристика
	Кількість головних гравців, од	2
	Загальний обсяг продаж, грн/ум.од	2 грн/ум.од
	Динаміка ринку (якісна оцінка)	Зростає
	Наявність обмежень для входу (вказати характер обмежень)	Недискримінаційні якісні
	Специфічні вимоги до стандартизації та сертифікації	Відсутні
	Середня норма рентабельності в галузі (або по ринку), %	69,5%

Таблиця 6.5 - Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Бажання зменшення часу роботи програміста та приріст швидкості реалізації додатків	1. Малий середній та великий бізнес галузі ІТ	1. Ведення бізнесу на платформах інтернет-ресурсів	1. Простий інтерфейс користувача 2. Можливість детального налаштування 3. Гнучкі ціни

Таблиця 6.6 - Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Крадіжка інтелектуальної власності	Крадіжка ідеї або ключової інтелектуальної інновації	Відсудження прав інтелектуальної власності Забезпечення якіснішого захисту інформації Зміна методики шифрування приватного ключа Попередження користувачів із подальшою співпрацею для мінімізації фактору загрози
2	Відмова компаній у співпраці	Керівництво компанії не погоджується на співпрацю	Пропозиція більш вигідних умов співпраці
3	Недостача стартових капіталовкладень	Недостача початкових інвестицій для реалізації мінімально життєздатного продукту	Пошук нових джерел інвестицій

Таблиця 6.7 - Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Отримання необхідних інвестицій	Сформований початковий капітал, необхідний для реалізації мінімально життєздатного продукту	Розробка мінімально життєздатного продукту
2	Співпраця з відомим компаніями які надють послуги оренди автомобілів	Керівництво відомих IT компаній вигідно використовувати даний продукт через скорочення затрат тачасовий приріст в рамках створення нових систем	Підтримка стабільної роботи системи та проведення масштабування системи
3	Висока зацікавленість користувачів	Обіг використання веб-служби становить більше 1000 запитів в день	Підтримка стабільної роботи системи та проведення масштабування системи Збільшення цін на використання сервісу

Продовження таблиці 6.7

4	Успішна маркетингова політика	В результаті проведеної маркетингової політики отримана висока зацікавленість користувачів	Підтримка стабільної роботи системи та проведення масштабування системи Збільшення цін на використання сервісу Використання подібної маркетингової стратегії надалі для залучення нових користувачів
5	Ліквідація конкурента	Конкурент ліквідував свою компанію у результаті власного бажання або зовнішніх чинників	Проведення маркетингової кампанії для монополізації ринку

Таблиця 6.8 - Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
Олігополія	Незначна кількість конкурентів Велика ринкова сила Схожість використовуваних технологій	Інформування ринку щодо появи нової веб-служби Співпраця із провідними сервісами
Галузевий	Загроза появи нових конкурентів Ринкова влада споживачів Висока потреба у товарі	Інформування ринку щодо якості використовуваної новаторської технології Пропозиція гнучких цін
Внутрішньогалузева	Діяльність в одній галузі економіки Надання сервісів одного типу	Зменшення вартості сервісу Примноження каналів розподілу
Товарно-видова	Надання різних сервісів одного виду	Маркетингова політика
Цінова	Використання цін для покращення економічних умов збуту	Зменшення вартості сервісу Використання нових каналів розподілу

Продовження таблиці 6.8

Марочна	Пропозиція схожого сервісу Спільна цільова аудиторія	Інформування ринку щодо якості використовуваної новаторської технології Примноження каналів розподілу

Таблиця 6.9 – Аналіз конкуренці в галузі за М. Портером

	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Складові аналізу	«WebSystem Generator», «Wordpress», «SimplCommerce »	Розмір капіталовкладень, Забезпечення гнучких цін, Доступ до каналів розподілу, Витрати на масштабах	Відсутні	- Змінні витрати: Виробничі непрямі дегресивні - Системи інформації: пропаганда, реклама та директ-маркетинг, - Рівень чутливості до цін: споживачі орієнтовані	Копіювання функціоналу, Монополізація дистриб’юторів, Мініміза ція цін

	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
				на цінність продукту - Продуктова диференціація: якість, спосіб отримання сервісу, швидкість обслуговування Методи контролю якості: тестування та профілювання, прототипування, інспектування коду, аналіз архітектури програмног	

	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
				о забезпечення	
Висновки	CR4 = 95% Індекс Херфіндаля-Хіршмана (HHI) = 6518 Значення показників вказує на високу концентрацію (монополізацію) даного ринку	Можливості входу на ринок забезпечить мінімізація цін, швидкість та простота надавання послуги споживачам і співпраця із головними гравцями ринку. В результаті аналізу проектів на народно-громадських інтернет-платформах потенційних конкурентів	Відсутні	Клієнти диктують умови гнучкості цінової політики, високої і довгострокової якості послуг та наявність кооперації із сервісами, що вони використовують	Пропонування вигідних умов дистриб'юторам, забезпечення захисту інтелектуальної власності, гнучкість цінової політики

	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
		знайдено не було			

Таблиця 6.3 - Порівняльний аналіз сильних та слабких сторін «назва проекту»

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з Avis						
			–3	–2	–1	0	+1	+2	+3
1	Унікальність сервісу	18							+
2	Модель бізнес для бізнесу	18							+
3	Цінова політика	10					+		
4	Додаткові послуги	18							+

Таблиця 6.4 - SWOT- аналіз стартап-проекту

Сильні сторони: Якість та довго тривалість послуги Низькі ціни Додаткові сервіси	Слабкі сторони: Недостача стартових капіталовкладень Бізнес модель залежить від політики окремих бізнесів Необхідність стрімкого входу на ринок
Можливості: Інвестиції Реалізація бізнес-моделі Реалізація бізнес-моделі Висока зацікавленість цільової аудиторії	Загрози: Крадіжка інтелектуальної власності Відмова у співпраці компаній Недовіра користувачів

Таблиця 6.12 - Альтернативи ринкового впровадження стартапів

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Створення власної компанії	Йовірне	12 місяців
2	Маркетингова кампанія для приваблювання користувачів	Ймовірне	2 місяці
3	Пропонування безкоштовних послуг	Ймовірне	4 місяці
4	Пошук бізнесів іншої галузі для співпраці	Мало цмовірне	4 місяців
Обрана альтернатива: Створення власної компанії			

6.4 Розробка ринкової стратегії проекту

Таблиця 6.13 Вибір цільових груп потенційних споживачів

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку*
1	Надання сервісу постачальникам продукції у якості веб-служби	Вибірковий розподіл	Здатність протистояти прямим конкурентам Низькі витрати	Стратегія диференціації

Таблиця 6.5 - Визначення базової стратегії розвитку

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачі в сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Малі компанії та приватні особи	Висока	83%		Високі бар’єри входу
2	Великі компанії	Середня	65%		Високі бар’єри входу
3	Середні компанії	Висока	75%		Низьки бар’єри входу
Які цільові групи обрано: Малі компанії та приватні особи					

Таблиця 6.15 - Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки*
1	Ні	Забирати та залучати нових	Ні	Стратегія лідера. Розширення первинного попиту

Таблиця 6.16 - Визначення стратегії позиціонування

п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
	Правельність та якість згенерованого коду Простий інтерфейс користувача Можливість налаштовувати параметри для кожного рівня додатку Гнучкі ціни Оперативний зв’язок з компанією, яка надає послуги оренди.	Стратегія диференціації	Формування регулярного попиту Збільшення разового використання послуги Виявлення нових груп споживачів Нові напрями застосування існуючої послуги	Генерація коду з економією робочих годин розробника. Навигідніший варіант. Простота використання

6.5 Розробка маркетингової програми стартап-проекту

Таблиця 6.17 - Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
1	Попит на зменшення часових рамок на виконання задачі	Гарантоване спрощення роботи, що економить час Зручний інтерфейс Низькі ціни	Якість послуги Інноваційність підходу Цінова перевага

Таблиця 6.18 - Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
	4500 – 10000 грн	6000-10000 грн	80000 грн	4500 – 10000 грн

Таблиця 6.19 - Формування системи збуту

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
	Ведення бізнесу в інтернеті	Прямі неофіційні	Послідовність в реалізації обраної позиції Доступність та об’єктивність інформації про фірму і товар	Формування у цільової аудиторії обізнаності про появу нового сервісу	Раціоналістична стратегія реклами

Таблиця 6.20 - Концепція маркетингових комунікацій

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
	Закупівля здійснюється через довірені джерела	Інформування користувачів	Канал одного рівня	Селективна з використанням комбінованого каналу збуту

ВИСНОВКИ

У дисертації розглянуто проблема швидкого створення ефективних web-додатків одного класу. Виконаний аналіз проблеми і існуючих рішень підтвердив актуальність проблеми і дозволив сформулювати її постановку, визначити ключові аспекти дослідження.

Запропоновано комплексний підхід до розв'язання проблеми, побудований на автогенерації шаблонних рішень для інформаційних систем з можливістю інтеграції в процес компонентів на різних архітектурних рівнях. Основними перевагами запропонованого рішення є формальна основа для створення додатків яке базується на шаблонізації процесів проектування та реалізації та можливість детального налаштування даного процесу. До того ж прийнята модель побудови шаблонних рішень спрощує роботу розробників та допомагає автоматизувати базовий процес розробки web-додатків широкого для інформаційних систем для бізнесу з web-представленням.

Аналіз проблеми та існуючих рішень надав можливість вибору відповідних формальних засобів. Математичний апарат для запропонованого рішення включає засоби семантичного описання бізнес-процесів та формальну логічну систему. Перші забезпечують описання бізнес-процесів і уможливлюють їх декомпозицією на підпроцеси від самого верхнього рівня архітектури до атомарних компонентів, що не піддаються декомпозиції, з вказанням вхідної та вихідної сутності та характеристик за рахунок яких проводиться вибір відповідної функціональної одиниці – шаблону для опрацювання підпроцесу, або створення власного шаблону за рахунок вибору запропонованих методів. Формальна логічна система полегшує вибір і інтеграцію компонентів в комплесне рішення.

Також розглянута програмна реалізація описаної проблеми. Наведено її загальний опис, аргументовано архітектуру технлогії, порівняно та обрано програмні засоби для її реалізації. Розглянуто на прикладі інтерфейс системи та її роботу.

Подальші дослідження пов’язані з 3D-візуалізацією процесів створення web-додатків на основі запропонованої теоретичної моделі для побудови комплексного рішення по шаблону процесу створення інформаційної системи з web-представленням.

ПЕРЕЛІК ПОСИЛАНЬ

1. Sergii Telenyk, Grzegosz Nowakowski, Kostyantyn Yefremov, Volodymyr Khmelyuk. Logics based application integration for interdisciplinary scientific investigations // 2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 21-23 Sept. 2017, Bucharest, Romania. DOI: 10.1109/IDAACS.2017.8095241

2. Шаховська Н.Б. Методи опрацювання консолідованих даних за допомогою просторів даних / Н.Б.Шаховська // Проблеми програмування. – 2011. – № 4. – С. 72-84.

3. Текущая спецификация WSDL [Електронний ресурс] Режим доступу до журн.: http://www.w3.org/tr/wsdl

4. Текущая спецификация UDDI [Електронний ресурс] Режим доступу до журн.: http://uddi.org/pubs/uddi_v3.htm

5. Теленик С.Ф. Семантична інтеграція різнорідних інформаційних ресурсів / С.Ф.Теленик, О.А.Амонс, К.В.Ефремов, С.В.Жук // Вісник НТУУ «КПІ», Інформатика, управління та обчислювальна техніка». – №58. –2013. – С.29 – 45.

6. Теленик С.Ф. Логічний підхід до інтеграції програмних застосувань підтримки міждисциплінарних наукових досліджень // С.Ф.Теленик, О.А.Амонс, К.В.Єфремов, В.Т.Лиско // Наукові вісті НТУУ «КПІ». –№5 (91). –2013. – С.53–72.

7. Теленик С.Ф. Управління високопродуктивними IT-інфраструктурами / Ю.В.Бойко, М.М.Глибовець, С.В.Єршов, С.Л.Кривий, С.Д.Погорілий, О.І.Ролік, С.Ф.Теленик, М.В.Ясочка // Вісник НТУУ «КПІ», Інформатика, управління та обчислювальна техніка». – №61. –2014 . – С.120 – 141..

8. Джеффри Рихтер CLR via C# / Рихтер Джеффри. – Санкт-Петербург: Питер, 2011. – (ISBN 978-5-7502-0348-2).

9. Node.js[Електронний_ресурс]_–_Режим_доступу_до_ресурсу: https://uk.wikipedia.org/wiki/Node.js.

10. _База_даних[Електронний_ресурс]_Режим_доступу_до_ресурсу:_https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85.

11. Криспин Л. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд / Л. Криспин, Д. Грегори., 2010. – 464 с. – (Вильямс). – (ISBN 978-5-8459-1625-9).

12. Управління ресурсами даних [Електронний ресурс] – Режим доступу до ресурсу: https://sites.google.com/site/webvemon/iste-3-1.

13. Cacti - The Complete RRDTool-based Graphing Solution [Електронний ресурс] – Режим доступу до ресурсу: http://www.cacti.net.

14. Cacti (software) [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Cacti_(software).

15. Let’s Encrypt [Електронний ресурс] – Режим доступу до ресурсу: https://letsencrypt.org/.
16. Oracle. Siebel Business Process Framework: Workflow Guide Version [Електронний ресурс] – Режим доступу до ресурсу: http://docs.oracle.com/cd/B40099_02/books/PDF/BPFWorkflow.pdf.
17. HP Service Activator (HPSA) solution development training [Електронний ресурс] – Режим доступу до ресурсу: http://www8.hp.com/h20195/v2/GetPDF.aspx%2F4AA4-9576ENW.pdf.
18. NetCracker. The Solution Products [Електронний ресурс] – Режим доступу до ресурсу: http://www.nec.com/en/global/techrep/journal/g10/n02/pdf/100221.pdf.
19. Репин В.В. Процессный подход к управлению. Моделирование бизнес-процессов – М.: РИА «Стандарты и качество», 2004.
20. ISO 9000:2000 – Quality management systems.
21. Information technology – Object Management Group Business Process Model and Notation / ISO/IEC 19510, 2013.
22. K. Heather. Navigating the SOA Open Standards Landscape Around / The Open Group, 2009. – 27p.
23. Web Services Architecture – W3C Working Group Note [Електронний ресурс]– Режим доступу до ресурсу: http://www.w3.org/TR/ws-arch/.
24. SOA Reference Architecture. – The Open Group / ISBN: 1-937218-01-0, 2011. – 192 p.
25. Reference Architecture Foundation for Service Oriented Architecture Version.— The Organization for the Advancement of Structured Information Standards (OASIS), 2012 [Електронний ресурс] – Режим доступу до ресурсу: http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra.html.
26. Service-Oriented Architecture Ontology. – The Open Group / ISBN: 1-931624-88-7, 2010.

ДОДАТКИ

Додаток А. Публікації.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНСТИТУТ МОДЕРНІЗАЦІЇ ЗМІСТУ ОСВІТИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

WINTER INFOCOM ADVANCED SOLUTIONS 2018

МАТЕРІАЛИ

VII МІЖНАРОДНОЇ НАУКОВО-ПРАКТИЧНОЇ КОНФЕРЕНЦІЇ

З ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

CONFERENCE PROCEEDINGS

7th SCIENTIFIC AND PRACTICAL CONFERENCE

КИЇВ, УКРАЇНА

1-2 грудня 2018 року

2 грудня	***Інформаційні системи та технології***	
	Ременюк Д. **Долина В.**	Системи локального позиціонування для управління транспортними засобами на гірському підприємстві
	Кирилюк А. **Долина В.**	Механізми боротьби з колізіями при використанні технології радіочастотної ідентифікації
	Звіряка В. **Долина В.**	Аналіз джерел альтернативного енергоживлення для тепличних господарств
	Kharabet R. **Pysarenko A.**	Information System for Household Goods Accounting
	Poltorak V.	Locators Field for Effective BCH code
	Alhawawsha M.	The Role and Challenges of E-government in Jordan and USA
	Технології програмування	
	Озеракін М. **Амонс О.**	Алгоритм консенсусу для систем збереження на основі технології блокчейн
	Дорошенко А. **Туліка Є.**	Еквівалентні перетворення послідовних програм до моделі акторів
	Шатов О. **Вовк Є.** **Амонс О.** **Хмелюк В.**	Масштабована система розпізнавання осіб для спостереження за об'єктами

Масштабована система розпізнавання осіб для спостереження за об'єктами

Шатов Сергій
КПІ ім. Ігоря Сікорського
Київ, Україна
uristmegollor@gmail.com

Вовк Євгеній
КПІ ім. Ігоря Сікорського
Київ, Україна
vovkzenia@gmail.com

Амоюс Олександр
КПІ ім. Ігоря Сікорського
Київ, Україна
amous@ex.ua

Хмелюк Володимир
КПІ ім. Ігоря Сікорського
Київ, Україна
hmelyuk@gmail.com

Анотація. **Впродовж великого часу автоматизовані системи відеоспостереження за об'єктами розвивалися шляхом впровадження все більш ефективних алгоритмів виявлення руху. З появою нових алгоритмів виявлення руху на основі застосування технологій машинного навчання, збільшення продуктивності графічних процесорів в умовах виходу технологій з відкритим вихідним кодом на етап зрілості розвиток автоматизованих систем відеоспостереження отримує новий імпульс. Нові технології, засоби обчислювальної техніки і вдосконалені алгоритми дозволяють будувати автоматизовані системи відеоспостереження з високими параметрами. В роботі описана нова побудована на сучасних технологіях система відеоспостереження, параметри якої дозволять використовувати її для широкого класу об'єктів. Основна увага приділена аспектам багатофункціонального розпізнавання осіб на основі машинного навчання з вивченням метрик.**

Ключові слова: обробка зображень, хмарний шар, високопродуктивні обчислення, спостереження, машинне навчання, розпізнавання осіб.

Вступ

Існує низка об'єктів, наприклад навчальні та оздоровчі заклади, режимні об'єкти, в яких необхідно відслідковувати переміщення персоналу, учнів та інших осіб, відпрацьовуючи певні політики стосовно їх поведінки. З іншого боку, на цих та інших об'єктах виникає потреба відслідковувати появу осіб, які ще невідомі системі.

Для підтримки функціонування таких об'єктів потрібна побудована на новітніх технологіях система відеоспостереження, яка була б не надто дорогою, простою в експлуатації та надійною. У кількісному відношенні особливу цікавість становлять системи, які вимагають відслідковування відеопотоків до 100 відеокамер, що використовуються для фіксації. Постає цікава науково-практична проблема, вирішення якої вимагає нового поєднання відомих ідей.

У роботі пропонується оригінальне рішення, в основу якого покладено нову схему взаємодії трьох технологій, яка дозволила зберегти переваги кожної з них і нівелювати недоліки, тим самим досягти синергетичного ефекту. Сильними сторонами розробленої системи є здатність підтримувати велике навантаження, виділяти обличчя і тулуб, розпізнавати осіб, будувати базу даних траєкторій та вже знайомих людей. Зрештою це дозволяє створювати і контролювати зони доступу, подавати сигнал тривоги у випадку їх порушення, наприклад при проникненні на територію незнайомої особи чи перетину особою меж певної зони.

Для оцінки якості роботи були використані дві метрики – максимальна кількість паралельних відеопотоків, які можуть бути оброблені без покращення часу реагування, та ймовірність розпізнавання особи за 2 секунди, протягом яких її зображення присутнє на відео.

Пов'язані роботи

Високопродуктивні обчислення. Розроблена система використовує декілька технологій високопродуктивних обчислень (high-performance computing). Щоб опрацьовувати одночасно від 50 до 100 відеопотоків одночасно, в системі підтримуються паралельні обчислення, реалізація яких виконана на основі пакетів лінійних обчислень LAPACK (який використовується через бібліотеку numpy у мові python) та CUDA, брокера повідомлень RabbitMQ, баз даних (файлових, реляційних, in-memory).

Розпізнавання осіб. Одночасне опрацювання потоків великої кількості відеокамер стало можливим завдяки використанню розвинених технологій розпізнавання та лінійних обчислень на відеокартах. Ці технології широко використовуються у системі та мають визначальне значення для досягнення вдалого відношення якості роботи системи до швидкості її роботи. У роботі нижче будуть наведені порівняльні характеристики деяких нейромереж, для визначення продуктивності яких використовувалися формули, запропоновані у праці [2].

Машинне вивчення метрики. Для виконання високих вимог до швидкості системи використовуються сучасні технології машинного навчання. Найкращі результати показало вивчення метрик – напрям машинного навчання, у якому робиться спроба не класифікувати або кластеризувати приклади, а збудувати модель, спроможну їх відрізнити [3].

Оброблення відеопотоку

Відеопотік з камер може оброблятися в двох режимах, вибір яких здійснюється персоналом. Незалежно від вибору режиму оброблення, відеопотік буде проіндексовано і збережено в файловій базі даних. Але вибір режиму оброблення впливає на наповнення БД новою інформацією. При будь-якому режимі роботи, в першу чергу відео підлягає перед-обробленню на відеокарті з метою виявлення наявності людей на кадрах.

Перед-оброблення. Відеопотік розбивається на окремі кадри, і кожен n-й (значення n встановлюється) кадр обробляється нейромережею. Результатом роботи є виділені прямокутники, кожен з яких з визначеною ймовірністю вказує на виявлену людину всередині виділеного простору. Значення n залежить від ресурсів системи, відеонавантаження і вимог по швидкодії. Ці ж чинники впливають також на вибір моделі перед-оброблення. Нижче буде наведена таблиця результатів дослідження декількох моделей з відкритим кодом, наданих Google в рамках бібліотеки Tensorflow, за кількістю вибору кадрів в секунду і максимальною кількістю паралельних відеопотоків в секунду.

Режим розпізнавання траєкторій. Траєкторії будуються за допомогою математичної моделі (1), яка враховує параметри камер, їх розташування у просторі і природу переміщення людей:

$$x = K[Rt]X \quad (1)$$

де x – точка на камері, K – матриця внутрішніх параметрів камери, R і t – матриці трансформації повороту і переміщення відповідно (матриці розташування камери в просторі), X – точка простору, причому обидві точки якісні в гомогенних координатах.

На основі моделі (1) будується "зворотна математична модель камери" (2), яка і дозволяє провести промінь з камери, виходячи з координат точки на зображенні:

$$X = [Rt]^{-1}K^{-1}x \quad (2)$$

Для розпізнаних осіб траєкторії будуються по кадрам. Центри прямокутників використовуються як центроїди, які пересуваються по відео разом з рухом пікселів. Відповідно центроїди людей є не тільки на перед-оброблених кадрах, а й на всіх інших. Після цього за допомогою зворотної моделі камери траєкторії трансформуються з простору кадрів на просторові координати. Для повного вирішення рівняння зворотної проекції і обчислення не тільки кута положення особи на площині відносно камери, але і її координат приймається обґрунтоване припущення, що у кожній зоні люди знаходяться на площині, яка має певну однакову для всіх висоту.

Режим розпізнавання осіб. Верхні частини прямокутників перевіряються з використанням інших моделей на наявність обличь. Це може бути прості відомі моделі на зразок HOG, або ж інші більш складні моделі на основі нейронних мереж. Всі виявлені обличчя конвертуються у вектори в метричному просторі. Для цього використовується ще одна нейронна мережа, яка навчається на датасеті осіб. Виходячи з сформульованих вище вимог до системи відеоспостереження кожна особа повинна бути представлена в датасеті кілька разів. Тому обрана модель типу metric learning, яка побудована не на категоризації/сегментації, а оцінки певних метрик для різних прикладів особи в датасеті. У випадку системи відеоспостереження оцінюється метрика належності двох обличь одній особі. Обличчя з кадрів зберігаються в БД відповідно до правила, яке враховує унікальність і корисність зображення обличчя особи. Відповідно, з продовженням роботи, в БД збирається все більше добре розпізнаних зображень осіб і модель покращує якість розпізнавання вже знайомих осіб, паралельно навчаючись розпізнавати ще незнайомих [2].

Архітектура системи

Загальний вигляд роботи системи в штатному режимі необхідно розділити на дві складових фонові задачі та обробка та дії за вимогою клієнту. Фонова обробка включає в себе отримання відео потоку з камер та його подальшу передачу через шину даних на сервіс по його збереженню в історії та сервіс по його опрацюванню, після успішного збереження відповідний запис з'являється в БД, а саме запис що містить інформацію по одній людині та запис що містить історію переміщень людей пов одну камеру, результат опрацювання відео потоку також зберігається в БД, відповідно з вказаним людей що були знайдені, не відомих системі людей, після чого відповідна інформація передається на вихідну точку системи.

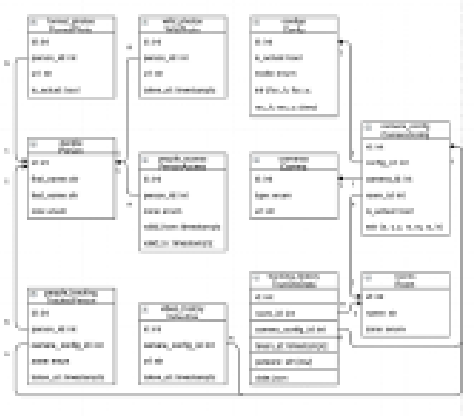

Рис. 1. Схема бази даних

БД, схема якої наведена на рис. 1, містить інформацію про співробітників (анкету співробітника, перелік приміщень, в які йому дозволений доступ, та час, зображення співробітника та результат їх опрацювання нейронною мережею – відповідно таблиці person, person_access, wild_photo, formal_photo). Також БД включає: групи таблиць, в яких описані кімнати та камери: характеристики камер (кут огляду і т.п.), камери та їх тип і адреси для доступу та отримання відеопотоку; розв'язуюча таблиця, у якій подано точне місце розташування кожної камери у відповідній кімнаті; таблиця опису кімнат (характеристики кімнат, місця їх розташування). Для збереження результатів оброблення відео з метою пришвидшення роботи системи запропоновано дві оригінальні сутності. Перша з них використовується для збереження інформації по конкретній особі і надання можливості відслідковування її траєкторії, а друга – для надання можливості відображення траєкторій усіх осіб в зоні однієї камери.

Експериментальне обґрунтування

Для оцінювання результатів бінарної класифікації використовуються часто вживані чотири значення: вірний позитивний (TP – true positive), вірний негативний (TN – true negative), хибний позитивний

(FP – false positive); хибний негативний (FN – false negative).

Для оцінювання моделі використовуються показники влучності P, відкликання R та точності A, значення яких розраховуються за допомогою відповідно формул (3), (4) і (5).

$$P = \frac{TP}{TP + FP} \quad (3)$$

$$R = \frac{TP}{TP + FN} \quad (4)$$

$$A = \frac{TP + TN}{TP + TN + FP + FN} \quad (5)$$

Для оцінювання сегментації використовуються такі поширені показники як "передбачена зона об'єкту" (PA –prediction area) і "вірна зона об'єкту" (TA – true area). Для оцінювання якості розпізнавання окремого об'єкту можна використовувати показник перетину над об'єднанням (Intersection over Union), який розраховується за допомогою формули (6). Для оцінювання якості розпізнавання на датасеті використовується показник середньої влучності (mean Average Precision) на датасеті, який розраховується на основі відомої формули (7) [1].

$$IoU = \frac{PA \cap TA}{PA \cup TA} \quad (6)$$

$$mAP = \frac{1}{n}\sum_{i}^{i} \max_{k} (P) \quad (7)$$

Значення mAP = 30 означає що модель буде розпізнавати об'єкти з середньою влучністю 30%. Інакше кажучи, об'єкт буде коректно розпізнаний з імовірністю 30%, якщо він присутній на зображенні.

Для практичного оцінювання якості моделі відповідно до сформульованих вище вимог можна використовувати оригінальну формулу (8) розрахунку показника "шанс розпізнавання об'єкту у відеопотоці упродовж двох секунд" (2 seconds Recognition Probability) на основі значень показників середньої влучності та фіксованого значення F кадрів розпізнавання у секунду. Цей показник можна інтерпретувати як імовірність розпізнавання об'єкта у відеопотоці упродовж двох секунд.

$$2sRP = 1 - (1 - mAP)^{2F} \quad (8)$$

Нижче наведено табл. 1 з порівняльними оцінками різних моделей, включно з комплексною метрикою оцінювання якості роботи.

Таблиця 1

Порівняння моделей машинного навчання та їх характеристика

Архітектура	Характеристика				
	Швидкість (ms)	*Точність (mAP%)*	*Частота (frame/s)*	*Пропускна здатність (streams/s)*	*Імовірність розпізнавання (2sRP%)*
SSD MobileNet	31	22	2	16.1	63.0
SSD MobileNet	31	22	8	4	98.1
SSD Inception	42	24	2	11.9	66.6
SSD Inception	42	24	8	3	98.7
Faster RCNN Inception	58	28	2	8.6	73.1
Faster RCNN Inception	58	28	8	2.1	99.5
Faster RCNN ResNet50	89	30	2	11.2	76.0
Faster RCNN ResNet50	89	30	8	2.8	99.7
Faster RCNN ResNet101	106	32	1	9.4	53.7
Faster RCNN ResNet101	106	32	4	2.35	95.4

Дія таблиці дозволяє визначити якість різних моделей для практичного використання, та обрати потрібну конфігурацію у разі використання системи. Варто також зазначити, що метрика на основі оцінювання імовірності знаходження особи має сенс тільки у випадку, якщо особа займає положення, наприклад, проходить через кімнату.

Режим розпізнавання людини. Модель f, яка буде розпізнавати обличчя має оптимізувати функції (9) та (10). Вона приймає два приклади, які можуть належати або не належати до одного класу. Перша формула каже, що якщо обидва приклади з одного класу, то модель має видавати 1, та 0 у супротивному випадку [2, 3].

$$f(x \in X, y \in Y) \to 1, X = Y \quad (9)$$

$$f(x \in X, y \in Y) \to 0, X \neq Y \quad (10)$$

Модель яка використовується у системі має точність біля 99% на коректне розпізнавання обличчя у випадку тестового датасету на 100 людей, але вона значно погіршується на знімках або зображеннях фотографіях.

Режим побудови траєкторій. Побудова траєкторій не є складною справою, адже їх треба проводити тільки кожні пів секунди, поміж кадрами с розпізнаванням. Але після цього треба визначити чи два відрізки траєкторій належать до одної людини або двох різних. Для цієї задачі використовується модель, схожа до моделі першого режиму роботи, але конструктивно вона може мати іншу архітектуру [3]. Наприклад, можна використовувати фільтр Кальмана.

Висновки

Розроблена система відеоспостереження, що задовольняє вимогам, поставленим у вступі вимогам. Запропонована архітектура системи, вибір алгоритмів і технологій реалізації забезпечують її ефективне використання на підприємствах середнього розміру. При цьому розробку характеризують висока надійність роботи, розумна стійкість до масштабності будь якого з внутрішніх серверів. Система може бути масштабована до потреб з різним балансом якості та вартості, за експериментальними даними, що наводиться у роботі у таблицях порівняльних характеристик різних моделей.

Литература

1. Lin, Tsung-Yi et al. "Microsoft COCO: Common Objects in Context." Lecture Notes in Computer Science (2014): 740–755.

2. Zheng, Liang et al. "Person Re-Identification in the Wild." 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017).

3. Xing, Ng et al. "Distance metric learning, with application to clustering with side-information" NIPS'02 Proceedings of the 15th International Conference on Neural Information Processing Systems.

The 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications
18-21 September, 2019, Metz, France

Conceptual foundations of the use of formal models and methods for the rapid creation of web-applications

Sergii Telenyk [1], Nowakowski Grzegorz [2], Eduard Zharikov [3], Vovk Jewhenii [4]
[1] Department of Theoretical Electrical Engineering and Computer Science, Faculty of Electrical and Computer Engineering, Cracow University of Technology, Cracow, Poland, stelenyk@pk.edu.pl, https://www.pk.edu.pl
[2] Department of Automatic Control and Information Technology, Faculty of Electrical and Computer Engineering, Cracow University of Technology, Cracow, Poland, gnowakowski@pk.edu.pl, pk.edu.pl
[3,4] Department of Automation and Control in Technical Systems, National Technical University of Ukraine "Igor Sikorsky Kyiv, Polytechnic Institute", Kyiv, Ukraine, [3]zharikov.eduard@acts.kpi.ua, [4]kafedra@acts.kpi.ua, kpi.ua/en

***Abstract*—The problem of the rapid creation of effective web-applications of one class with using formal means are considered. The conceptual approach to its solution is offered on the basis of analysis of the peculiarities of web-applications construction. The approach is based on defining the standard web application architecture and selecting its components using formal methods in accordance with user requirements. A formal logical system is used, which uses the design of web-applications as a process of outputting the formula specified according to the needs of the user, which defines the schemes of execution of the modules of the system. An important feature of the approach is the ability to 3D-visualize the process of designing the system, which creates conditions for effective interaction between developers and machine development tools.**

Keywords— web-applications, application construction, business processes, mathematical logic

I. Introduction

The creation of information systems today is based on modern methodological concepts that inherited the most important ideas of classical methodologies such as SADT, IDEF0, while enriching them with new ideas.

The most important directions for improving classical methodologies were: (1) structuring software that opened the way for programmer's teamwork and reuse of software code; (2) describing the created system possibilities using languages of conceptual modeling, which was accompanied by the development of automated design systems; (3) structuring the design process, which enriched the industry with the concept of the system life cycle, repository and unified models of role design systems; (4) prototyping, on the basis of which the concept of parallel development of parts of the system, design and implementation of the system by parts were developed.

By and large, modern methodologies for information systems design, for example Agile-methodology, only successfully use different combinations of these concepts, filling them with real content and complementing the elements of psychology.

Formalization is very specific in modern methodologies [1]-[4]. In classical methodologies it was considered as the basis of automation, CASE-tools. Today, formalization is more related to describing the architecture of the system and descriptions are mainly used to set tasks for project participants, discuss intermediate results, define the next steps [1]-[2].

The article attempts to use the developed formal models and effective methods for building the technology of automated generation of software systems on the basis of modern methodologies.

As this problem has enormous scales, we have identified one of the most advanced components of modern information systems - the development of systems with web-presentation.

II. The Essence Of The Problem

The expediency of this tasks class automation is explained by the considerable complexity of design and implementation of subtasks, to which they decompose. Subproblems are duplicated in each project, and changes are subject to data but not their main transformation.

Developers should agree on a universal architecture. Then the application will have the same overall appearance and differ from others only by the presence or absence of one or another component depending on the functional requirements for application.

Adhering to the principles of the three-tier architecture [5], in the typical application we will allocate levels - representation, business logic and access to data. The level of data access determines the ability to work with data and in general, it is the implementation of the template *Repository*. The access to data, if necessary, from the business logic we define in the traditional way: 1) with each entity, we associate certain generalized behavior,

inherent in all elements of the level of access to data; 2) expanding behavior by adding unique methods for the current entity.

Accordingly, a specific repository class inherits its interface, obtaining basic methods common to all and its own methods from its own interface, and also inherits the class that implements the basic behavior, the fact of which contains the implementation of only their methods. The template has a specific architecture and is expanded by adding new classes and interfaces to work with each entity.

Business logic [5] is responsible for processing data obtained from the level of access to data, through the use of behavioral patterns, such as strategy, behavioral classes, in certain cases of commands. Each component implements the end-user functionality that the user expects. According to the incorporated concept, components can be expanded by adding classes with narrowly oriented behavior.

The *Strategy* template aims to implement a set of different behaviors depending on the needs of the component that is level higher and appeals to this strategy. A strategy is described with an appropriate method that takes on input the required parameters, and several implementations of the method.

The *Command* template provides the implementation of a certain behavior on demand, allowing abstract from the logic of the command itself inside the mechanism.

Another mechanism that is used is an interface with behavior descriptions and a implementation class that contains the elements of data processing and the formation of the result required for a component that is in the architecture of the level above.

A web-presentation is a system component that is accessible from the outside and visible to all users. It is intended for receiving and analyzing queries and performing necessary actions by means of calling business logic methods, as well as displaying data on a client side. This component is mainly built on an MVC template, consisting of representations and directly methods invoked when processing client requests.

As the MVC template is built according to its purpose: the model determines which data should be displayed; representation is a graphical component capable of displaying a particular model; the controller determines how to get the model, which presentation to submit it and how to process the request.

By following the description, the task of creating any web-application can be considered as a task of filling in a particular template. But building even a template of correct architecture, writing monotone and long code require a lot of time. Therefore, the problem of automating the creation of a software system through the automated implementation of the system template based on user requirements. Solving this problem involves the following steps: designing a database model [5] and generating access levels to it; realization of data processing by built-in strategies on the user's request; generation of presentation. Therefore, the problem of automating the creation of a software system appears by automated implementation of the system template based on user requirements. Solving this problem involves the following steps: designing a database model and generating access levels to it; realization of data processing by built-in strategies on the user's request; generation of presentation.

III. Existing Approaches To Creating Web-Based Applications

Attempts to automate the creation of web-applications are made a long time ago.

Of course, it's worth mentioning already made solutions like CMS. The logic of the behavior of these constructors is to use the finished system for the entire range of tasks at the expense of the means of choosing exclusively the theme of the graphic design and adding or disconnecting those or other pages. This entails a number of shortcomings.

Another solution is the composite automation of the capabilities of individual modules through the use of frameworks built on one or another technology.

The article considers other approaches to automating the creation of web-applications and concludes that they are partially compliant with the solution to the problem of generating the code of web-applications taking into account the possibility of managing the automation process at each level and the appropriate capability of selecting and integrating existing components based on formal user requirements and descriptions of these components.

For more detailed consideration of the architecture of web-applications, an appropriate alternative to their rapid creation is to template the process of their implementation at each level of the architecture using existing frameworks developed for this or that technology.

But there are several obvious aspects of solving this problem. The level of access to data is subject to templating based on the entities of the database as a separate component. Similarly, you can customize the creation of a REST API [6] based on process descriptions.

The automation of the rapid creation of web-applications in the case of this approach leaves programmers one of the key issues - combining the resulting fragments into a general application, requiring appropriate changes to the autogenerated product code.

IV. Conceptual Foundations For Creating Web-Applications Based On Templates

The key idea behind the development is to build a real model based on the basic notion of architecture, to model the application development process.

set of template processes can be implemented with one application at the expense of means of using the frameworks and specifications of one or another technology. The application is multilevel and consists of

template modules. Each module is constructed of classes and interfaces that have their own structural units and the corresponding appearance and methods depending on the tasks to be solved in accordance with the requirements of users.

The design of any system begins with the implementation of business processes and the structure of the data warehouse, which will meet the objectives. The user has to put the schema model for the database - to describe the entities and indicate their links.

After this stage, the system builds the basic functionality in the form of a data model of scripts for database generation at the level of data access, using scenarios for work with users and authorizations that the user can choose depending on the needs.

The next stage of the user's work is the generation of the data component directly, namely data selection, data preparation with a number of conditions by combining conditions using the proposed template or creating your own.

After this stage, you need to set the conditions and create a semantic description of the corresponding transformation and processing of data in accordance with the current business process.

The final stage is to generate the presentation with the appropriate selection and editing the most suitable template from the proposed.

The result is a ready-made application, generated on the basis of the composition of the generated modules of the system

V. Mathematical Model For Constructing Basic Functionality

To select and integrate components in the complete solution, we will use the first-order causal logic, the structural elements of which are described by the authors for the integration of applications in the work [1].

Symbols:

service: (,), [,], {, }, :, <, >, ,;

constant:

- individual, of primary types (int, real, char, bool) - a_1^1, a_2^1, ..., a_1^2, a_2^2, ... where each constant a_i^k pertains to type (primary type) k; structural type (construct) - c_1, c_2, ...; procedural type (method) - d_1, d_2, ...; objective type (problem, entity, relation) — e_1, e_2, ...;
- *functional i-place*, for individuals of type k - h_1^1, h_2^1, ..., h_1^2, h_2^2, ...;
- *predicate i-place*, for individuals of type k - A_1^1, A_2^1, ..., A_1^2, A_2^2, ... (this class includes taxonomic, relational and other predicates, as well as traditional relations, at least equality = and order ≥);

variable: for individuals of type k - x_1^1, x_2^1, ..., x_1^2, x_2^2, ..., where every variable x_i^k pertains to type k;

logical: $\neg, \wedge, \vee, \leftarrow, \exists, \forall, \Leftrightarrow$.

Individual terms of type k:

- each individual constant a_i^k of type k is an individual term of type k;
- each free variable x_i^k for individuals of type k is an individual term of type k;
- if h_i^j is a certain functional constant for individuals of type k and τ1, ..., τj are terms for individuals of type k, then h_i^j (τ1, ..., τj) is an individual term of type k;
- there are no other individual terms of type k.

Formulas for individuals:

- if A_i^j is a predicate constant for individuals and τ1, ..., τj are terms for them, then A_i^j (τ1, ..., τj) is the atomic formula for individuals;
- the atomic formula for individuals is the formula for them;
- there are no other formulas for individuals.

Hereinafter we consider only the systems of Horn clauses (with a single → symbol, atomic formulas to its left and right, and an implicit quantifier $\forall$).

Specifier - it is construction of kind τ1, where τ1 is a therm for an individual object. The specifiers of construct:

- if $e_1^e \ldots e_i^e$, - individual object of entity kind, $e_1^r \ldots e_i^r$, - individual object of relation kind, $A_i^j(a_1^1 \ldots a_j^1) \ldots A_i^j(a_1^k \ldots a_j^k)$ - atomic formulas for the individuals of primary types, τ – individual therm of kind construct, then τ: ($e_1^e \ldots e_i^e$, $e_1^r \ldots e_i^r$, A_i^j ($a_1^1 \ldots a_j^1$) ... $A_i^j(a_1^k \ldots a_j^k)$) - specifier of construct;
- other specifiers of construct are not.

Precondition:

- if c_1 – specifier of construct, Π - sequence of atomic formulas, then <τ_1:(c_1,Π)> - elementary precondition;
- elementary precondition - precondition;
- if <τ_1> - precondition, τ_2 - elementary precondition, then <τ_1, τ_2> - precondition;
- other preconditions are not.

Post-condition:

- if τ_1 – specifier of construct, Π - sequence of atomic formulas, then <τ_1:(c_1,Π)> - elementary post-condition;
- elementary post-condition - post-condition;
- if <τ_1> - post-condition, τ_2 - elementary post-condition, then < τ_1, τ_2> - post-condition;
- other post-conditions are not.

Specifier of methods:

- if τ - individual therm of method kind, <τ_1> - precondition, < τ_2> - post-condition, then τ: (<τ_1>, < τ_2>) – specifier of method;
- other specifiers of methods are not.

Specifier of problems:

- if <τ_1> - precondition, a <τ_2> - post-condition, τ – therm for individual object of kind Problem, then τ: << τ_1>,< τ_2>> - specifier of problems;
- other specifiers of problems are not.

Clause - an expression of kind $\Pi \Rightarrow \Lambda$, where Π - sequence of atomic formulas; Λ - only atomic formula.

The system's knowledge base consists of the three main parts. In the first part, ontological axioms describe the methods and other components of the framework available for use. The second part provides an ontology of the system, described using OWL-based languages based on RDF. The third part summarizes the output rules needed to get the desired result for the user. There are rules for the output of two types. The first type output rules are used to integrate many methods and other components into a single application. They take into account the peculiarities of the problem, the preconditions, the stages, the descriptions of methods and other components of the system. Realized with the desired output proof in a certain way defines the architecture of the application that is being created within a defined class of architectures.

Inference rules of first type:

- If d1: (<τ1>,< τ2>) and d2: (<τ3>,< τ1>) then d1: (<d2>,< τ2>)
- If d1: (<τ1>,< τ2>) and d2: (<τ3 ^ τ4>,< τ1>) then d1: (<d2>,< τ2>)
- If d1: (<τ1 ^ τ2>,< τ3>) and d2: (<τ4>,< τ1>) and d3: (<τ5>,< τ2>) then d1: (<d2 ^ d3>,< τ3 >)
- If d1: (<τ1>,< τ2>) and d2: (<τ3 ∨ τ4 >,< τ1>) then d1: (<d2>,< τ2>)
- If d1: (<τ2 ^ τ2>,< τ1>) then d1: (<τ2>,< τ1>)
- If d1: (<τ1, τ2>,< τ3>) and d2: (<τ4>,< τ1>) and d3: (<τ5>,< τ2>) then d1: (<d2, d3>,< τ2>)
- If d1: (<τ1>,< τ2,τ3>) and d2: (<τ2>,< τ4>) and d3: (<τ3>,< τ5>) then d2: (<d1>,< τ4>) and d3: (<d1>,< τ5>)

The second type inference rules are used to speed up the inference process. They take into account the semantics of the user's problem and the semantics of methods and other components of the system, expressed in terms of the system's ontology. In essence, these rules are used to reduce the search for rules and axioms of the first part of the knowledge base of the system. They describe semantically acceptable at the level of input and output the variants of combining methods and other components of the system in more functionally complete combinations. These combinations can be used to deduce in the space of rules of the first type. The meaningful use of the second type inference rules can be illustrated by the process of 3D visualization of the combination of methods and other components of the system in spatial structures based on common entities on their inputs / outputs.

VI. The Semantically Controlled Inference Mechanism

For the inference we will use the approach proposed by the authors in the paper [1]. It's about the procedures for inference and restoring the inference tree. But we will add to this approach the preliminary procedure for searching the spatial structure of the associated by input/output methods and other components of the system, which at inputs has entities defined by the user as input information, and at its outputs we have the entities defined by the user as the source information. Define the necessary concepts.

Naturally, we focus on the means of knowledge representation and to reduce the search, we use the knowledge base, especially information on effective inference schemes for frequently performed queries and its ability to structure, factorize and abstract. This is a combined inference strategy, at the lower levels of which the birth of a lossy resolvents is blocked. Formally it is a question of combining the approach of R. Kowalski [3] and the analogue method of D. A. Plaisted [4]. In this case, the proof in the abstract space, the result of which is then used to control the inference in the initial search space, is based exclusively on the axioms and inference rules of the knowledge base.

We cut off the hopeless branches of the inference in the initial space in two stages. At the first stage, we formulate the constructions of the associated by input/output methods and other components of the system. At the second stage, we use the inference in the abstract space. For the reflection of the transition to the abstract space, we will use taxonomic connections. Then the inference in the abstract space will be reduced to the inference in the system of types (classes of entities) with a gradual deepening of the system of subtypes up to individuals. As before, to reduce the search in both the abstract (for classes and types) and in the initial (for individuals) spaces, we will use modifications to the Robinson resolution strategy. The properties needed for the mutual adaptation of the method of analogy and modification of the resolution strategy the used reflection do naturally acquire in a responsibly structured knowledge base.

For the analogy method, we use the concept of a multi clause (*m*-clause) as a multiset of atomic formulas (atomic formula *L* will be recorded in *m*-clause as many times as it is repeated). The operations $\cup$ (union), $\cap$ (intersection), — (difference), · (concatenation) and relation $\subseteq$ (occurrence) for multisets are naturally performed (note that the operations are performed for the left and right parts of the clause separately).

Suppose $A_1 \subset C_1$, $A_2 \subset C_2$ and α_1, α_2 are substitutions, which allow us to obtain the most common unifier for atomic formulas A_1 and A_2. Then the clause obtained from unification of clauses $C_1\alpha_1$ and $C_2\alpha_2$ by removing L to the left and right of symbol $\rightarrow$ is called *m*-resolvent of *m*-clauses C_1 and C_2. If the *m*-clauses C_1 and C_2 are ordered and *m*-resolvent is obtained by eliminating the non-underlined atomic formula in both parts, and on the left after it there is no other atomic formula, then we get the ordered linear *m*-resolvent. If the last atomic formula to the left of the symbol $\rightarrow$ of the ordered *m*-clause is unified with the underlined atomic formula to the right of the symbol $\rightarrow$ of the same *m*-clause the ordered linear *m*-resolvent is obtained by the *m*-clause reduction.

Definitions of *m*-clauses and *m*-resolvents are used to define the ordered linear *m-resolutional* proof. Let's start with the definition of the *m-resolutional* proof *Tm* as a pair of <*V*, *Th*>, where *V* is the set of proof vertices, *Th* is the set of vertices triples. In the future, the first and second components of *Tm* will be allocated using the *s-N(Tm)* and *s-M(Tm)* selector functions respectively. Each vertex $n \in$ *s-N(Tm)* of the proof *Tm* is characterized by the mark *s-L(n)* and the depth *s-D(n)*. If $<n_1, n_2, n_3> \in$ *s-M(Tm)*, then *s-L*(n_3) is the *m*-resolvent *s-L*(n_1) and *s-L*(n_2), and each triple of this type is called *m*-resolution. In the proof *Tm*, the vertex $n \in$ *s-N(Tm)* is called the initial if it is not the third component of any of the triples of *s-M(Tm)* (its mark is the initial *m*-clause), or terminal if it is not neither the first nor the second component of any of the triples of *s-M(Tm)* (its mark is the terminal *m*-clause).

Now, let's call the *m-resolutional* proof *Tm* the proof from S, if the marks of the initial vertices *Tm* belong to the set of *m*-clause S. From S we deduce C if *Tm* is a proof from S, and C is a mark of one of the vertices of *Tm*.

Finally, the ordered linear *m-resolutional* proof from S is called *m-resolutional* proof *Tm* from S, all *m*-clauses of which are ordered and for an arbitrary triple $<n_1, n_2, n_3>$ from *s-M(Tm)* *s-L*(n_3) is an ordered linear *m*-resolvent *s-L*(n_1) and *s-L*(n_2).

To manage an ordered linear inference, we will use the typification abstraction proposed in [4]. Suppose f is such mapping from the set of *m*-clauses into the set of *m*-clauses that: 1) if *m*-clause C_3 is the *m*-resolvent of *m*-clauses C_1 and C_2, while $D_3 \in f(C_3)$, then exist such $D_1 \in f(C_1)$ and $D_2 \in f(C_2)$ that the result of the substitution of some m-resolvent D_1 and D_2 belongs to D_3; 2) $f(\varnothing) = \{\varnothing\}$; 3) if the result of some substitution of *m*-clause C_1 belongs to *m*-clause C_2, then for any abstraction D_2 for C_2 exists such abstraction D_1 for C_1 that the result of D_1 substitution belongs to D_2. Such mapping is called f *m*-abstraction mapping, while any D from $f(C)$ is called *m*-abstraction. The typification mapping is understood as a certain mapping ϕ from a set of literals into a set of literals, which reflects each atomic formula into the formula whose terms have the type closest to the basic types in the hierarchy.

Construction of the proof begins with the formulation of the problem by the user. In essence, the possibility of executing the request is checked, taking into account all available resources. Having access to the descriptions of all modules and other components and axioms that determine the possibilities of their application, the inference mechanism combines the modules and other components into a structure (proof), which ensures the receipt of the result from the input data.

Let the ordered linear *m-resolutional* proof be obtained by definition as a pair of sets: vertices $V = \{k_1, k_2, k_3, \ldots k_n\}$ and vertices triple $Th = \{<k_1, k_2, k_3>, <k_3, k_4, k_5> \ldots <k_{n-2}, k_{n-1}, k_n>\}$, where k_i, $i = 1,\ldots, n$ are the vertices of proof, k_n is the terminal vertex. The proof is formed, taking on the initial vertices of the problem postcondition and, as a lateral vertex, the appropriate axiom from the knowledge base. The axiom is appropriate if the sequence of atomic formulas corresponding to this vertex of the proof, or any part of it, is an axiom postcondition. The third vertex of this triple is obtained by applying the axiom to the formula postcondition, taking into account the inference rules. The third vertex of the first triple becomes the first vertex of the second triple, and the process repeats until the terminal vertex is reached - the problem precondition. If the atomic formula corresponding to the third vertex of the triple can be simplified by applying one of the inference rules, it is simplified in the next triangle, which will have only two vertices - a vertex with a formula, which must be simplified, and a vertex with a simplified formula.

The search for the corresponding axioms is done by comparing the construct of the postcondition, which is being processed at the moment, with the axioms constructs from the ontology. Thus, from the ontology a set of axioms is chosen that describe the methods that process the type and format of the objects we need.

If in the knowledge base for the next vertex the corresponding axioms are several, then each of these axioms is used to further build its own "parallel" version of the proof. Thus, during the operation of the inference mechanism, a number of proofs of varying length and complexity can be formed. Upon completion of the inference mechanism work, the proof from the set of proofs is selected with the smallest number of triples of vertices and the smallest number of applied axioms.

To shorten the search, we can pre-select the structures of related methods and other components, the entities of which inputs and outputs are common. And only in the second stage, when the structures are already selected, the preconditions and postconditions are checked and the architecture of the solution is determined.

A set of formulas and a clause that corresponds to the vertices of the proof triples and reflects the application of the axiom of the first part and the inference rules of the

first type is the output of the inference mechanism. To restore the structure, we use the proof tree recovery algorithm proposed by the authors in the paper [1].

Then, using the constructor, we are generating by the finished elements the user representation, to which, in the form of a data source, the output points of the business logic of the application are tied.

VII. Implementation Of Tecihnology

The technology is realized using modern tools. The user-friendly web-interface allows the end-user to build a database model based on which the base framework of the application is generated (CRUD data operations with the subsequent generation of the interface as a REST API [6]). The next step is to display this general view of the application with the ability to generate by means of a semantic assignment of relationships between the relevant models of one or another application logic behavior and implementation of the business processes of the required application.

Ready templates are integrated into the application with pointing inputs and outputs. Additional functionality is the ability to create and store by user own templates as well as the ability to put them in the public domain. Formally, the template integration can take place at different levels of architecture, ranging from method integration to integration of entire component that consists of classes.

The applications are a composition of modules, which, in turn, are a composition of classes, which, in turn, are a composition of methods, etc. Defining by the user of input and output entities and their characteristics allows you to find the necessary methods and their interconnections and get the template model of the lowest architectural level, presented in the form of classes.

At the highest level, the composition model is maintained, but the corresponding classes are already merged through the calls of other methods and the generation of behavior classes, which is the source of the module. Accordingly, the combination of modules follows from the composition of their outgoing points and the established connections.

VIII. Conclusion

In the article the problem of the rapid creation of effective web-applications of one class is considered. The performed analysis of the problem and existing decisions confirmed the relevance of the problem and allowed formulate it, identify the key aspects of the study.

The complex approach to the solution of the problem, based on autogenous template solutions for information systems with the possibility of integration into the process of components at different architectural levels, is proposed. The main advantages of the proposed solution are the formal basis for creating applications, which is based on templating the design and implementation processes and the ability to fine-tune the process. In addition, the adopted model for constructing template solutions simplifies the work of developers and helps automate the basic process of developing web-applications for a wide range of business information systems with web-representation.

Analysis of the problem and the existing solutions enabled the choice of appropriate formal means. The mathematical apparatus for the proposed solution includes means of semantic description of business processes and a formal logical system. The first provides a description of business processes and enables them to decompose subprocesses from the highest level of architecture to non-decomposable atomic components, indicating the input and output entity, and the characteristics at which the choice of the corresponding functional unit is made - a template for the processing of the subprocess, or creation own template by choosing the proposed methods. A formal logic system facilitates the selection and integration of components in a complete solution.

Further researches are connected with 3D-visualization of the processes of creating web-applications on the basis of the proposed theoretical model for building a comprehensive solution based on the template of the process of creating an information system with a web-representation.

ACKNOWLEDGMENT

Presented results of the research, which was carried out under the theme No. E-3/586/2018/DS, were funded by the subsidies on science granted by Polish Ministry of Science and Higher Education.

References

[1] S. Talaryk, O. Nowakowski, K. Yatbarov and V. Khmelnik, *Logics based application integration for interdisciplinary scientific investigations*, 2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Bucharest, 2017, pp. 1026-1031. DOI: 10.1109/IDAACS.2017.8095241

[2] A.Y. Levy, *Logic-Based Techniques in Data Integration*, In: Logic Based Artificial Intelligence, Edited by J. Minker. Kluwer Publishers, 2000. DOI: 10.1007/978-1-4615-1567-8_24

[3] R. Kowalski, *Computational logic and human thinking: how to be artificially intelligent*. Cambridge University Press, 2010. DOI: 10.1017/CBO9780511984747

[4] D. A. Plaisted, *History and prospects for first-order automated deduction*. International Conference on Automated Deduction. Springer, Cham, 2015. DOI: 10.1007/978-3-319-21401-6_1

[5] O. Nowakowski, *Open source relational databases and their capabilities in constructing a web-based system designed to support the functioning of a health clinic*, Technical Transactions, vol. 1-AC/2013, 53–65.

[6] O. Nowakowski, *Rest Api safety assurance by means of JWT mechanism*. Information Systems in Management, 2016, Vol. 5, No. 3, s. 359-369.

Додаток Б. Лістинг коду типової веб-системи.

Абстракція бізнес-логіки

```
using System.Threading.Tasks;
using CommandInvoke.Entities.Ui.CommandViewModel;

namespace CommandInvoke.Abstract.Bl.CommandProcessing
{
    public interface ICommandProvider
    {
        Task<string>  WriteCommandAsFile(CreateCommand  command,string
userId);
        Task<bool> SaveIntoDb(CreateCommand command, string userId);

        Task<string> BaseProcessing(CreateCommand command, string userId);
    }
}
using System.Threading.Tasks;

namespace CommandInvoke.Abstract.Bl.Common
{
    public interface IEmailSender
    {
        Task SendEmailAsync(string email, string subject, string message);
    }
}
```

Імплементація бізнес-логіки

```
using System;
```

```
using System.IO;
using System.Threading.Tasks;
using System.Transactions;
using CommandInvoke.Abstract.Bl.CommandProcessing;
using CommandInvoke.Dal.Abstract;
using CommandInvoke.Dal.Abstract.Repositories;
using CommandInvoke.Db.Entities;
using CommandInvoke.Entities.Ui.CommandViewModel;
using CommandInvoke.Entities.Ui.Configurations;
using CommandInvoke.Entities.Ui.Enums;
using Microsoft.Extensions.Options;
using Newtonsoft.Json;
using DbExtAbstrc = CommandInvoke.Dal.Abstract;
using DbExtImpl = CommandInvoke.Dal.Impl.Ef;

namespace CommandInvoke.Bl.Impl.CommandProcessing
{
   public class CommandProvider : ICommandProvider
   {

      private readonly FileSaveParams _configuration;
      private readonly ICommandRepository _commandRepository;
      private readonly IUserCommandRepository _userCommandRepository;
      private readonly DbExtImpl.ApplicationDbContext _context;
      public CommandProvider(IOptions<FileSaveParams> config,
         ICommandRepository commandRepository,
         IUserCommandRepository userCommandRepository,
         DbExtImpl.ApplicationDbContext context)
      {
         _configuration = config.Value;
```

```
            _commandRepository = commandRepository;
            _userCommandRepository = userCommandRepository;
            _context = context;
        }
        public async Task<string> WriteCommandAsFile(CreateCommand command, string userId)
        {
            SaveFileType enumVal;

            bool parseRes =
                Enum.TryParse(_configuration.SaveType, out enumVal);

            if (!parseRes)
            {
                enumVal = SaveFileType.none;
            }

            string path = String.Empty;

            //Here shud be factory if we had few ways to save file
            switch (enumVal)
            {
                case SaveFileType.none:
                    //TODO: Log
                    break;
                case SaveFileType.json:
                    string output =
                        JsonConvert.SerializeObject(command);
                    path = await SaveFile(output, userId, "json");
                    break;
```

```
            case SaveFileType.txt:
                throw new NotImplementedException();
            default:
                throw new ArgumentOutOfRangeException();
        }

        return path;
    }

    protected async Task<string> SaveFile(string text, string userId,string format)
    {
        string path = _configuration.SaveDirectory;

        if (!Directory.Exists(path))
        {
            Directory.CreateDirectory(path);
        }

        if (_configuration.SaveAtUserDirectory)
        {
            path = Path.Combine(path, userId);

            if (!Directory.Exists(path))
            {
                Directory.CreateDirectory(path);
            }
        }
```

```
            path = Path.Combine(path, $"{Path.GetRandomFileName()}.{format}");

            //Here can be logic for throw exception or other handling data
            //if (File.Exists(path))
            //{
            //    File.Delete(path);
            //}

            using (FileStream fs = new FileStream(path, FileMode.OpenOrCreate))
            {
                using (StreamWriter sw = new StreamWriter(fs))
                {
                    await sw.WriteAsync(text);
                }
            }

            return path;
        }

        public async Task<bool> SaveIntoDb(CreateCommand command, string userId)
        {
            using (DbExtAbstrc.Common.ITransactionScope scope =
                new DbExtImpl.Common.TransactionScope(_context))
            {
                try
                {
                    Command cmd = new Command()
                    {
```

```
                    Name = command.CommandName
                };
                await _commandRepository.AddAsync(cmd);
                await _commandRepository.SaveAsync();

                UserCommand userCmd = new UserCommand()
                {
                    CommandId = cmd.Id,
                    CreatedTime = DateTimeOffset.Now,
                    UserId = userId
                };
                await _userCommandRepository.AddAsync(userCmd);
                await _userCommandRepository.SaveAsync();

                scope.Commit();
                return true;
            }
            catch
            {
                //TODO : log
                scope.RollBack();
                return false;
            }
        }
    }

    public async Task<string> BaseProcessing(CreateCommand command, string userId)
    {
        await this.SaveIntoDb(command,userId);
```

```
            return await this.WriteCommandAsFile(command, userId);
        }
    }
}
using System.Threading.Tasks;
using CommandInvoke.Abstract.Bl.Common;

namespace CommandInvoke.Bl.Impl.Common
{
    // This class is used by the application to send email for account confirmation
and password reset.
    // Here shud be yours implementation of send logic
    public class EmailSender : IEmailSender
    {
        public Task SendEmailAsync(string email, string subject, string message)
        {
            return Task.CompletedTask;
        }
    }
}
```

Абстракція рівня доступу до даних

```
using System;
using System.Collections.Generic;
using System.Linq.Expressions;
using System.Threading.Tasks;

namespace CommandInvoke.Dal.Abstract
{
    public interface IGenericKeyRepository<TKey, TEntity>
```

```
        {
            Task<TEntity> AddAsync(TEntity entity);
            Task UpdateAsync(TEntity entity);
            Task<TEntity> DeleteAsync(TEntity entity);
            Task<List<TEntity>> GetAllAsync();
            Task<List<TEntity>>  GetByAsync(Expression<Func<TEntity, bool>>
predicate);
            Task<TEntity> GetByIdAsync(TKey id);
            Task<int> GetCountAsync();
            Task<int>        GetCountAsync(Expression<Func<TEntity,        bool>>
predicate);
            Task<List<TEntity>> FetchAsync();
            Task<List<TEntity>> FetchByAsync(Expression<Func<TEntity, bool>>
predicate);
            Task<List<TEntity>> PaggingFetchAsync(int startIndex, int count);
            Task<TEntity>  FirstOrDefaultAsync(Expression<Func<TEntity, bool>>
predicate);
            Task<List<TEntity>> PaggingFetchByAsync(Expression<Func<TEntity,
bool>> predicate,
                int startIndex, int count);
            Task SaveAsync();
        }

        public        interface        IGenericRepository<TEntity>        :
IGenericKeyRepository<int,TEntity>
        {

        }
    }
```

```
using System.Collections.Generic;
using System.Threading.Tasks;
using CommandInvoke.Db.Entities;
using CommandInvoke.Entities.Ui.AdminAreaViewModels;

namespace CommandInvoke.Dal.Abstract.Repositories
{
   public interface IUserCommandRepository : IGenericRepository<UserCommand>
   {
      Task<List<AdminCommandDisplay>> GetUserCommandListAdmin();
   }
}

using CommandInvoke.Db.Entities;

namespace CommandInvoke.Dal.Abstract
{
   public interface IUserRepository : IGenericRepository<ApplicationUser>
   {
   }
}

using CommandInvoke.Db.Entities;

namespace CommandInvoke.Dal.Abstract
{
   public interface ICommandRepository : IGenericRepository<Command>
   {
   }}
```

Імплементація рівня доступу до даних

```
using CommandInvoke.Db.Entities;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace CommandInvoke.Dal.Impl.Ef
{
   public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
   {

      public virtual DbSet<Command> Commands { get; set; }
      public virtual DbSet<UserCommand> UserCommands { get; set; }

      public
ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
         : base(options)
      {
      }

      protected override void OnModelCreating(ModelBuilder builder)
      {
         base.OnModelCreating(builder);
         base.OnModelCreating(builder);
      }
   }
}

using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Linq.Expressions;
using System.Threading.Tasks;
using CommandInvoke.Dal.Abstract;
using Microsoft.EntityFrameworkCore;

namespace CommandInvoke.Dal.Impl.Ef.Repositories
{
    public class GenericKeyRepository<TKey, TEntity> : IGenericKeyRepository<TKey, TEntity> where TEntity : class
    {
        public GenericKeyRepository(ApplicationDbContext context)
        {
            Context = context;
        }

        public ApplicationDbContext Context { get; }

        public DbSet<TEntity> DbSet => Context.Set<TEntity>();

        public virtual async Task<TEntity> AddAsync(TEntity entity)
        {
            var item = await Context.Set<TEntity>().AddAsync(entity);
            return item.Entity;
        }

        public virtual async Task UpdateAsync(TEntity entity)
        {
            Context.Entry(entity).State = EntityState.Modified;
            await Task.FromResult(0);
        }
```

```
public virtual async Task<TEntity> DeleteAsync(TEntity entity)
{
   TEntity result = Context.Set<TEntity>()
      .Remove(entity).Entity;
   return await Task.FromResult(result);
}

public virtual async Task<List<TEntity>> GetAllAsync()
{
   return await Context.Set<TEntity>().ToListAsync();
}

public virtual async Task<List<TEntity>> GetByAsync
   (Expression<Func<TEntity, bool>> predicate)
{
   return await Context.Set<TEntity>().Where(predicate).ToListAsync();
}

public virtual async Task<TEntity> GetByIdAsync(TKey id)
{
   return  await Context.Set<TEntity>()
      .FindAsync(id);
}

public virtual async Task<int> GetCountAsync()
{
   return await Context.Set<TEntity>().CountAsync();
}
```

```
public virtual Task<int> GetCountAsync
   (Expression<Func<TEntity, bool>> predicate)
{
   return Context.Set<TEntity>().CountAsync(predicate);
}

public virtual async Task<List<TEntity>> FetchAsync()
{
   return await Context.Set<TEntity>().ToListAsync();
}

public virtual async Task<List<TEntity>> FetchByAsync
   (Expression<Func<TEntity, bool>> predicate)
{
   return await Context.Set<TEntity>().Where(predicate)
      .ToListAsync();
}

public virtual async Task<List<TEntity>> PaggingFetchAsync
   (int startIndex, int count)
{
   return await Context.Set<TEntity>().Skip(startIndex)
      .Take(count).ToListAsync();
}

public virtual async Task<TEntity> FirstOrDefaultAsync
   (Expression<Func<TEntity, bool>> predicate)
{
   return await Context.Set<TEntity>().FirstOrDefaultAsync(predicate);
}
```

```
        public virtual async Task<List<TEntity>> PaggingFetchByAsync
            (Expression<Func<TEntity, bool>> predicate, int startIndex, int count)
        {
            return await Context.Set<TEntity>().Where(predicate)
                .Skip(startIndex).Take(count).ToListAsync();
        }

        public Task SaveAsync()
        {
            return Context.SaveChangesAsync();
        }
    }

    public class GenericRepository<TEntity> : GenericKeyRepository<int,
TEntity>, IGenericRepository<TEntity> where TEntity : class
    {
        public GenericRepository(ApplicationDbContext context) : base(context)
        {
        }

        public override async Task<TEntity> GetByIdAsync(int id)
        {
            return id <= 0 ? null : await Context.Set<TEntity>()
                .FindAsync(id);
        }
    }
}

using CommandInvoke.Dal.Abstract;
```

```
using CommandInvoke.Db.Entities;

namespace CommandInvoke.Dal.Impl.Ef.Repositories
{
    public class CommandRepository : GenericRepository<Command>, ICommandRepository
    {
        public CommandRepository(ApplicationDbContext context) : base(context)
        {
        }
    }
}

using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using CommandInvoke.Dal.Abstract;
using CommandInvoke.Dal.Abstract.Repositories;
using CommandInvoke.Db.Entities;
using CommandInvoke.Entities.Ui.AdminAreaViewModels;
using Microsoft.EntityFrameworkCore;

namespace CommandInvoke.Dal.Impl.Ef.Repositories
{
    public class UserCommandRepository : GenericRepository<UserCommand>, IUserCommandRepository
    {
        public UserCommandRepository(ApplicationDbContext context) : base(context)
```

```
        {
        }

        public async Task<List<AdminCommandDisplay>>
GetUserCommandListAdmin()
        {
            IQueryable<AdminCommandDisplay> userCommand = (from uc in
Context.UserCommands
                                                           join command in
Context.Commands on uc.CommandId equals command.Id
                                                           join users in Context.Users on
uc.UserId equals users.Id
                                                           join userRoles in Context.UserRoles
on users.Id equals userRoles.UserId
                                                           join role in Context.Roles on
userRoles.RoleId equals role.Id
                                                           select new
AdminCommandDisplay()
                                                           {
                                                              UserName = users.UserName,
                                                              CommandName =
command.Name,
                                                              CommandId = command.Id,
                                                              UserId = users.Id,
                                                              RoleId = role.Id,
                                                              UserRole = role.Name,
                                                              UserCommandRecordId = uc.Id,
                                                              Created = uc.CreatedTime
                                                           });
            return await userCommand.ToListAsync();       }    }}
```

```
using CommandInvoke.Dal.Abstract;
using CommandInvoke.Db.Entities;

namespace CommandInvoke.Dal.Impl.Ef.Repositories
{

  public class UserRepository:GenericRepository<ApplicationUser>,
IUserRepository
  {
    public UserRepository(ApplicationDbContext context) : base(context)
    {
    }
  }
}
```

Proxy сутності по роботі з БД

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace CommandInvoke.Db.Entities
{
  public class UserCommand
  {
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    [Key]
    [Column("Id")]
    public int Id { get; set; }
```

```
        [ForeignKey("User")]
        public string UserId { get; set; }
        [ForeignKey("Command")]
        public int CommandId { get; set; }
        public ApplicationUser User { get; set; }
        public Command Command { get; set; }
        public DateTimeOffset CreatedTime { get; set; }
    }

}
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace CommandInvoke.Db.Entities
{
    public class Command
    {
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        [Key]
        [Column("Id")]
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
using Microsoft.AspNetCore.Identity;

namespace CommandInvoke.Db.Entities
{
    public class ApplicationUser : IdentityUser
    {    }}
```

Сутності для відображення користувацьких даних

```
using System;
using CommandInvoke.Entities.Ui.Shared;

namespace CommandInvoke.Entities.Ui.AdminAreaViewModels
{
   public class AdminCommandDisplay : CommandDisplay
   {
      //Display data
      public string UserName { get; set; }
      public string UserRole { get; set; }
      public DateTimeOffset Created { get; set; }
      // Need for edit
      public string UserId { get; set; }
      public string RoleId { get; set; }
   }
}
using CommandInvoke.Entities.Ui.Shared;

namespace CommandInvoke.Entities.Ui.CommandViewModel
{
   public class CreateCommand : CommandDisplay
   {

   }
}

using System.ComponentModel.DataAnnotations;
namespace CommandInvoke.Entities.Ui.AccountViewModels
{
```

```
    public class LoginViewModel
    {
        [Required]
        [EmailAddress]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }

        [Display(Name = "Remember me?")]
        public bool RememberMe { get; set; }
    }
}

using System.ComponentModel.DataAnnotations;
namespace CommandInvoke.Entities.Ui.AccountViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email")]
        public string Email { get; set; }

        [Required]
        [StringLength(100, ErrorMessage = "The {0} must be at least {2} and at
max {1} characters long.", MinimumLength = 6)]
        [DataType(DataType.Password)]
        [Display(Name = "Password")]
```

```
        public string Password { get; set; }

        [DataType(DataType.Password)]
        [Display(Name = "Confirm password")]
        [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]
        public string ConfirmPassword { get; set; }
    }
}
```